МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

КАФЕДРА ЕЛЕКТРОНІКИ, ІНФОРМАЦІЙНИХ СИСТЕМ ТА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

Кваліфікаційна робота

другий (магістерський)
(рівень вищої освіти)

на тему **Адаптивна комп'ютерна системи оцінювання знань студентів**

Виконав: студент 2 курсу, групи 8.1214-іпз-2
спеціальності 121 Інженерія програмного забезпечення
(код і назва спеціальності)
освітньої програми Інженерія програмного забезпечення
(код і назва освітньої програми)
Чистіков В.С.
(ініціали та прізвище)
Керівник доцент, к.ф.-м.н., доцент І.А.Скрипник
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)
Рецензент Директор ТОВ «Дісітел» .
П.О. Лютий
(посада, вчене звання, науковий ступінь, підпис, ініціали та прізвище)

Запоріжжя
2025

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ІНЖЕНЕРНИЙ НАВЧАЛЬНО-НАУКОВИЙ ІНСТИТУТ
ім. Ю.М. Потебні
ЗАПОРІЗЬКОГО НАЦІОНАЛЬНОГО УНІВЕРСИТЕТУ

Кафедра __електроніки, інформаційних систем та програмного__ забезпечення
Рівень вищої освіти _____другий (магістерський)_____
Спеціальність __121_Інженерія програмного забезпечення_____
(код та назва)
Освітня програма ___Інженерія програмного забезпечення
(код та назва)

ЗАТВЕРДЖУЮ

Завідувач кафедри_____Т.В. Критська
"_01_" квітня_2025_року

З А В Д А Н Н Я

НА КВАЛІФІКАЦІЙНУ РОБОТУ СТУДЕНТОВІ

Чистікову Вадиму Сергійовичу
(прізвище, ім'я, по батькові)

1. Тема роботи _Адаптивна комп'ютерна системи оцінювання знань студентів_____

керівник роботи__к.ф.-м.н., доцент Скрипник Ірина Анатоліївна_____
(прізвище, ім'я, по батькові, науковий ступінь, вчене звання)
затверджені наказом ЗНУ від 28.03.2025 р. № 361-е____
2. Строк подання студентом кваліфікаційної роботи___01.12.2025_____
3. Вихідні дані магістерської роботи
- комплект нормативних документів ;
- технічне завдання до роботи.

4. Зміст розрахунково-пояснювальної записки (перелік питань, які потрібно розробити)
- огляд та збір літератури стосовно теми кваліфікаційної роботи;
- огляд та аналіз існуючих рішень та аналогів;
- дослідження проблеми адаптивного оцінювання знань студентів та методів її реалізації;
- створення програмного продукту та його опис;
- перелік вимог для роботи програми;
- дослідження поставленої проблеми та розробка висновків та пропозицій.

5. Перелік графічного матеріалу (з точним зазначенням обов'язкових креслень)
_____слайдів презентації

6. Консультанти розділів магістерської роботи

Розділ	Прізвище, ініціали та посада консультанта	Підпис, дата
		Завдання прийняв

7. Дата видачі завдання 01.04.2025

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів магістерської роботи	Строк виконання етапів магістерської роботи	Примітка
1	Аналіз предметної області в галузі оцінювання знань та адаптивних освітніх систем	01.04-15.04.25	виконано
2	Формулювання основної задачі дипломної роботи та узгодження її з науковим керівником	16.04-18.04.25	виконано
3	Аналіз існуючих підходів до адаптивного тестування та систем E-learning	19.04-30.04.25	виконано
4	Дослідження методів побудови адаптивних питань та моделей складності	01.05-15.05.25	виконано
5	Вибір архітектури системи та інструментів реалізації	16.05-25.05.25	виконано
6	Узгодження технічного завдання та подальших дій із науковим керівником	26.05-28.05.25	виконано
7	Розробка структури бази даних (питання, рівні складності, результати студентів)	29.05-12.06.25	виконано
8	Реалізація модуля формування адаптивних тестів	13.06-05.07.25	виконано
9	Реалізація модуля аналізу відповідей і коригування рівня складності	06.07-25.07.25	виконано
10	Інтеграція модулів та попереднє тестування системи	26.07-15.08.25	виконано
11	Порівняння результатів роботи системи на тестових користувачах (імітаційні дані)	16.08-05.09.25	виконано
12	Реалізація користувацького інтерфейсу системи (веб/десктоп)	06.09-30.09.25	виконано
13	Оформлення пояснювальної записки та підготовка матеріалів курсової	01.10-01.12.25	виконано

Студент _____ Чистіков В.С.
(підпис) (прізвище та ініціали)

Керівник роботи _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

Нормоконтроль пройдено

Нормоконтролер _____ Скрипник І.А.
(підпис) (прізвище та ініціали)

АНОТАЦІЯ

Сторінок: 87

Рисунків: 3

Таблиць: 7

Джерел: 19

Чистіков В. С. Адаптивна комп’ютерна система оцінювання знань студентів : кваліфікаційна робота магістра спеціальності 121 «Інженерія програмного забезпечення» / наук. керівник І. А. Скрипник. Запоріжжя : ЗНУ, 2025. 87 с.

Мета і завдання дослідження полягають у вивченні сучасних підходів до автоматизованого та адаптивного оцінювання знань студентів, а також у створенні комп’ютерної системи, що динамічно підлаштовує складність тестових завдань відповідно до індивідуальних результатів користувача.

У процесі роботи були розглянуті методи адаптивного тестування, моделі оцінювання рівня знань та алгоритми формування персоналізованої траєкторії перевірки. На основі проведеного аналізу розроблено адаптивну комп’ютерну систему, здатну автоматично добирати складність питань, обробляти відповіді та формувати індивідуальні профілі студентів.

Фронтенд-частина системи реалізована у Flutter мовою Dart, що забезпечує кросплатформність та зручність взаємодії. Бекенд побудовано на Java з використанням Spring Boot, що дозволило створити масштабований серверний застосунок із REST API та підтримкою інтеграції з базою даних. Розроблена система забезпечує стабільну роботу, адаптивність процесу тестування та гнучке керування навчальним контентом.

Ключові слова: *адаптивне тестування, оцінювання знань, комп’ютерна система, Java, Spring Boot, Flutter, Dart, автоматизація навчання*.

SUMMARY

Pages: 87

Figures: 3

Tables: 7

Sources: 19

Chystikov V. S. Adaptive Computer System for Student Knowledge Assessment: bachelor's/master's qualification work in specialty 121 "Software Engineering" / scientific supervisor I. A. Skrypnyk. Zaporizhzhia : ZNU, 2025. 87 p.

The aim and objectives of the research are to study modern approaches to automated and adaptive knowledge assessment of students and to develop a computer system that dynamically adjusts the difficulty level of test tasks according to the individual performance of each user.

During the research, methods of adaptive testing, models for evaluating the level of knowledge, and algorithms for forming a personalized assessment trajectory were examined. Based on the conducted analysis, an adaptive computer system was developed, capable of automatically selecting question difficulty, processing user answers, and generating individual student profiles.

The frontend part of the system was implemented using Flutter with the Dart programming language, ensuring cross-platform compatibility and a user-friendly interface. The backend was developed in Java using the Spring Boot framework, which enabled the creation of a scalable server application with a modular architecture, REST API support, and integration with a database. The resulting system provides stable operation, adaptive testing capabilities, and flexible management of educational content.

Keywords: *adaptive testing, knowledge assessment, computer system, Java, Spring Boot, Flutter, Dart, educational automation*.

ЗМІСТ

ВСТУП **9**

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПІДХОДІВ ДО АДАПТИВНОГО ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ **14**

1.1 Огляд проблеми оцінювання знань у вищій освіті 14

1.2 Підходи до автоматизованого тестування студентів 16

1.3 Моделі адаптивного оцінювання 18

1.3.1 Теорія Item Response Theory (IRT) 21

1.3.2 Алгоритми підбору складності тестових завдань 22

1.3.3 Переваги та недоліки адаптивних систем 22

1.4 Аналіз сучасних систем тестування 23

1.4.1 Системи Moodle та їх адаптивні модулі 24

1.4.2 Комерційні адаптивні платформи 25

1.4.3 Інші рішення на ринку освітніх систем 25

1.5 Висновки з розділу 1 25

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ АДАПТИВНОЇ СИСТЕМИ **28**

2.1 Архітектурні підходи до побудови веб- та мобільних систем 28

2.2 Інструменти для розробки фронтенд-частини 29

2.2.1 Фреймворк Flutter 30

2.2.2 Мова Dart 30

2.3 Інструменти для бекенд-розробки 31

2.3.1 Мова Java 32

2.3.2 Фреймворк Spring Boot 32

2.3.3 Реалізація REST API 33

2.4 Засоби зберігання та обробки даних 33

2.4.1 Реляційні бази даних 34

2.5 Аналіз протоколів взаємодії клієнт-сервер 34

2.6 Висновки з розділу 2 35

РОЗДІЛ 3 РОЗРОБКА АДАПТИВНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ 37
3.1 Проєктування системи 37
3.1.1 Моделювання вимог 37
3.1.2 Модель предметної області 39
3.1.3 Архітектурна схема системи 42
3.2 Реалізація бекенд-частини 43
3.2.1 Налаштування середовища Spring Boot 44
3.2.2 Реалізація REST API для тестування 49
3.2.3 Алгоритми адаптивного підбору запитань 53
3.2.4 Модуль автентифікації та авторизації 55
3.2.5 База даних: структура і моделі 56
3.3 Реалізація фронтенд-частини 57
3.3.1 Налаштування середовища Flutter 58
3.3.2 Проєктування інтерфейсу користувача 61
3.3.3 Взаємодія з бекендом через API 63
3.3.4 Модуль тестування та відображення результатів 64
3.4 Інтеграція системи 65
3.4.1 Взаємодія клієнта та сервера 66
3.4.2 Тестування окремих модулів 67
3.4.3 Загальне тестування працездатності 68
3.5 Висновки з розділу 3 69
РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ АДАПТИВНОЇ СИСТЕМИ ТЕСТУВАННЯ 71
4.1 Аналіз роботи алгоритмів адаптивного підбору завдань 71
4.1.1 Порівняння з фіксованими тестами 71
4.1.2 Визначення стійкості адаптивності 73
4.1.3 Можливості покращення алгоритму 74
4.2 Оцінка продуктивності системи 76
4.2.1 Продуктивність бекенд-частини 76

4.2.2 Продуктивність фронтенд-частини 77
4.2.3 Аналіз використання ресурсів 79
4.3 Оцінка зручності використання 80
4.3.1 Зворотний зв’язок користувачів 80
4.3.2 UX-аналіз інтерфейсу 82
4.4 Висновки з розділу 4 83
ВИСНОВКИ 85
СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 86

ВСТУП

Актуальність теми

Сучасна система освіти переходить від традиційних форм контролю знань до цифрових рішень, здатних забезпечити індивідуальний підхід до кожного студента. Зі зростанням кількості дистанційних курсів та онлайн-платформ виникає потреба у створенні інструментів, які не лише перевіряють рівень підготовки, але й адаптуються до особливостей користувача. Одним із таких рішень є адаптивні комп'ютерні системи оцінювання знань.

Подібні системи дозволяють аналізувати відповіді студента в реальному часі, підбирати наступні завдання відповідно до рівня складності, формувати персоналізовані траєкторії навчання та надавати викладачам розширену аналітику. Важливість таких технологій зростає і через необхідність підвищення об'єктивності оцінювання, зменшення людського фактору та оптимізації навантаження на педагогів.

Сучасні технології, зокрема Flutter для кросплатформної розробки інтерфейсу та Java Spring Boot для створення високонавантажених бекенд-сервісів, дозволяють реалізувати ефективну та масштабовану адаптивну систему оцінювання знань. Тому дослідження та розробка такої системи є актуальним як з наукової, так і з практичної точки зору.

Мета і завдання дослідження

Метою дослідження є розробка адаптивної комп'ютерної системи оцінювання знань студентів, яка забезпечуватиме індивідуальне формування тестових завдань, автоматизований аналіз відповідей та зручний інтерфейс для викладачів і студентів.

Для досягнення мети були поставлені такі завдання:

1. Проаналізувати сучасні підходи та інструменти до адаптивного оцінювання знань.

2. Дослідити архітектури та технології, придатні для створення масштабованих освітніх систем.

3. Розробити структуру бекенд-сервісу на основі Java та Spring Boot.

4. Реалізувати кросплатформний інтерфейс користувача за допомогою Flutter (Dart).

5. Створити механізм адаптивного підбору питань залежно від рівня знань студента.

6. Забезпечити зручну систему управління тестами та перегляду статистики для викладачів.

7. Провести тестування системи та оцінити її ефективність.

Об’єкт та предмет дослідження

Об’єктом дослідження є процес комп’ютерного оцінювання знань студентів.

Предмет дослідження

Предметом дослідження є методи та програмні засоби побудови адаптивної системи оцінювання знань, що реалізована на основі Flutter та Spring Boot.

Методи дослідження

Для реалізації та дослідження системи використовуються такі методи:

1. Аналіз наукових джерел щодо адаптивного тестування та електронних систем навчання.

2. Аналіз архітектурних підходів до розробки клієнт-серверних веб та мобільних застосунків.

3. Проєктування програмного забезпечення, включаючи використання діаграм та шаблонів проєктування.

4. Методи об’єктно-орієнтованого програмування при реалізації бекенд-частини на Java.

5. Методи кросплатформної розробки для створення інтерфейсу на Flutter.

6. Методи обробки даних та побудови адаптивних алгоритмів, що визначають рівень складності наступних завдань.

7. Тестування функціональних компонентів та проведення експериментів щодо продуктивності системи.

Наукова новизна одержаних результатів

Наукова новизна роботи полягає у поєднанні сучасних алгоритмів адаптивного тестування з практичною реалізацією кросплатформної комп’ютерної системи, здатної динамічно змінювати складність тестових завдань у режимі реального часу. У дослідженні:

- запропоновано структуру адаптивного модуля, що поєднує моделі оцінювання рівня знань (IRT, алгоритми рівневого добору завдань) з механізмами реактивного аналізу відповідей;
- удосконалено підхід до формування індивідуальних освітніх траєкторій шляхом використання комбінованих критеріїв визначення поточного рівня студента;
- реалізовано архітектуру взаємодії фронтенду (Flutter, Dart) і бекенду (Java, Spring Boot), що забезпечує швидкий обмін даними, модульність, масштабованість та можливість інтеграції з навчальними платформами;
- розроблено методику автоматичного побудування персоналізованих профілів користувачів і вибору наступного рівня складності завдання.

Практичне значення роботи

Практичне значення роботи полягає у створенні адаптивної комп’ютерної системи оцінювання знань, яка може бути використана у навчальних закладах для автоматизації контролю знань студентів. Завдяки використанню Flutter система працює на різних платформах (Android, iOS, Web, Desktop), що забезпечує зручність доступу. Бекенд, реалізований на Spring Boot, гарантує стабільну роботу, високу продуктивність та можливість масштабування.

Система може бути інтегрована в LMS, використовуватися для проведення модульних робіт, швидкого контролю знань або формувального оцінювання. Реалізовані адаптивні алгоритми дозволяють підвищити якість та

об’єктивність оцінювання, а також формувати індивідуальні траєкторії навчання для студентів.

Апробація одержаних результатів

Основні результати дослідження були апробовані під час роботи над навчальними проєктами з програмної інженерії, а також у межах наукових обговорень на кафедрі програмної інженерії Запорізького національного університету. Ключові положення, архітектурні рішення та методи адаптивного оцінювання були представлені та обговорені на:

- студентських науково-технічних семінарах спеціальності 121 «Інженерія програмного забезпечення»;
- засіданнях наукових гуртків, присвячених сучасним підходам до автоматизації навчання та побудові освітніх IT-систем.

За результатами апробації були внесені уточнення у структуру адаптивного модуля та оптимізовані алгоритми добору тестових завдань.

Глосарій

Адаптивне тестування — метод оцінювання знань, за якого система автоматично змінює рівень складності завдань залежно від відповідей користувача.

Алгоритм добору складності — набір правил, за допомогою яких система визначає, яке питання подавати наступним, враховуючи результати попередніх відповідей.

Бекенд — серверна частина програмної системи, що обробляє дані, виконує бізнес-логіку та забезпечує взаємодію між клієнтом і базою даних.

Flutter — кросплатформний фреймворк для розробки мобільних, веб- та десктопних застосунків, що використовує мову Dart.

Індивідуальна траєкторія оцінювання — послідовність тестових завдань, сформована з урахуванням рівня підготовки та поточних результатів студента.

Модель IRT (Item Response Theory) — математична модель, що описує залежність між характеристиками студента та складністю тестових завдань для точного вимірювання рівня знань.

Персоналізований профіль користувача — структура даних, що містить інформацію про успішність, рівень підготовки та індивідуальні відмінності студента.

REST API — інтерфейс взаємодії між клієнтом і сервером, що використовує протокол HTTP та стандартизовані методи запитів.

Spring Boot — Java-фреймворк, який спрощує створення серверних застосунків, забезпечує автоматичну конфігурацію та модульну архітектуру.

Тестове завдання — структурована одиниця оцінювання, що містить питання, варіанти відповідей (або інший формат відповіді) та рівень складності.

РОЗДІЛ 1 ДОСЛІДЖЕННЯ ПІДХОДІВ ДО АДАПТИВНОГО ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ

1.1 Огляд проблеми оцінювання знань у вищій освіті

У дослідженні [1] проведено комплексний аналіз сучасних підходів до оцінювання знань студентів, окреслено існуючі труднощі традиційних методик та наведено технології, що дозволяють удосконалити процес перевірки навчальних результатів. Автор звертає увагу на ключові сфери, у яких автоматизовані комп'ютерні системи можуть суттєво підвищити якість та об'єктивність оцінювання.

Після широкого впровадження електронного навчання та інструментів дистанційної освіти постала проблема підвищення ефективності та гнучкості контролю знань. Традиційні тести із фіксованою складністю демонструють низку недоліків: вони не враховують індивідуальні особливості студентів, потребують багато часу на підготовку та не завжди забезпечують достатню точність вимірювання рівня компетентностей. Це призводить до необхідності пошуку технологічних рішень, здатних автоматично адаптувати навчальний контент до можливостей кожного користувача.

Автор дослідження підкреслює, що попри активну цифровізацію освітнього процесу, універсального підходу до створення системи, яка б забезпечувала високу точність оцінювання та водночас залишалася гнучкою й доступною для широкого використання, поки не існує. У результаті вимоги до таких систем повинні аналізуватися окремо для кожного навчального сценарію. Проблему ускладнюють два чинники:

- По-перше, кількість критеріїв до подібних систем є великою: від точності визначення рівня знань і швидкості обробки відповідей до масштабованості, простоти використання та низької вартості впровадження.
- По-друге, наявні технології охоплюють широкий спектр алгоритмів — від класичних тестів до моделей машинного навчання та

адаптивних підходів, що потребують складної інфраструктури або спеціалізованих ресурсів.

Виникає питання: чому саме у вищій освіті проблема якісного оцінювання знань є настільки актуальною? У дослідженні стверджується, що університети працюють у середовищі високої варіативності рівня підготовки студентів. Це створює труднощі для розробки уніфікованих тестових матеріалів, які однаково добре підходили б усім здобувачам освіти. Поширені способи перевірки знань виявляють низку обмежень, що заважають отримувати достовірні результати.

До основних проблем традиційних систем оцінювання відносять такі чинники:

Фіксована складність завдань, яка не відображає різний рівень підготовки студентів.

Обмежена об’єктивність, оскільки оцінювання часто залежить від людського фактору.

Низька точність вимірювання, особливо для студентів із проміжним рівнем знань.

Великі часові витрати на створення, перевірку та оновлення тестових матеріалів.

Відсутність персоналізації, через що студенти не отримують індивідуальної освітньої траєкторії.

Водночас низка особливостей цифрового освітнього середовища робить можливим підвищення ефективності оцінювання. Серед таких переваг виділяють:

Можливість створення компактних та чітко структурованих модулів тестування.

Стабільність умов проходження тесту, що мінімізує вплив зовнішніх факторів.

Автоматизовану обробку відповідей, яка забезпечує високу швидкість отримання результатів.

Наявність технічної інфраструктури — мережевих ресурсів, електронних журналів, цифрових платформ.

Можливість застосування моделей адаптивного тестування, які підлаштовують складність завдань до рівня студента.

У результаті автор дослідження доходить висновку, що проблема якісного та об’єктивного оцінювання знань у вищій освіті залишається відкритою, а існуючі підходи демонструють як переваги, так і суттєві обмеження. Саме тому все більшої актуальності набувають адаптивні комп’ютерні системи, що використовують сучасні технології для створення ефективного, точного та персоналізованого процесу оцінювання.

1.2 Підходи до автоматизованого тестування студентів

У сучасній вищій освіті автоматизоване тестування є одним із ключових інструментів оцінювання знань студентів [3]. Воно дозволяє підвищити швидкість перевірки, зменшити людський фактор та забезпечити об’єктивність результатів. Існує декілька підходів до реалізації автоматизованого тестування, які відрізняються за принципом роботи, технологічними рішеннями та рівнем адаптивності до здібностей студентів.

Перший підхід базується на класичних електронних тестах, які містять заздалегідь сформовану базу запитань. Такі системи зазвичай пропонують студентам однакові завдання з фіксованою складністю. Основними перевагами цього підходу є простота реалізації та відносно низькі витрати на створення системи. Проте він не враховує індивідуальні особливості студентів і може бути неефективним для вимірювання реального рівня знань, особливо в групах з широким діапазоном підготовки.

Другий підхід передбачає адаптивне тестування, коли система динамічно підбирає завдання відповідно до попередніх відповідей студента. Адаптивні системи оцінювання можуть використовувати різні алгоритми: від простих правил зміни складності тесту до моделей на основі ймовірнісного

оцінювання знань (Item Response Theory, IRT). Такий підхід дозволяє зменшити кількість питань, які потрібно пройти студенту, та одночасно отримати більш точну оцінку його рівня підготовки.

Третій підхід включає використання штучного інтелекту та машинного навчання для аналізу результатів тестів у реальному часі та формування індивідуальної освітньої траєкторії. Такі системи можуть не лише підбирати завдання відповідно до рівня студента, а й надавати рекомендації щодо повторення матеріалу або пропонувати додаткові навчальні модулі. Вони активно використовують сучасні технології веб-розробки, бази даних та фреймворки для створення фронтенд- і бекенд-частин системи (наприклад, Flutter/Dart для інтерфейсу користувача та Java/Spring Boot для серверної частини).Для наочності порівняння підходів до автоматизованого тестування студентів наведена таблиця 1.

Таблиця 1 – Порівняння підходів до автоматизованого тестування студентів

Підхід	**Принцип роботи**	**Переваги**	**Недоліки**	**Рівень адаптивності**
Класичні електронні тести	Фіксований набір запитань однакової складності для всіх студентів	Простота реалізації, низькі витрати	Не враховує індивідуальні особливості студентів, обмежена точність оцінювання	Низький
Адаптивне тестування (IRT та ін.)	Система підбирає завдання залежно від попередніх відповідей	Більш точне оцінювання рівня знань, зменшує кількість питань	Складність реалізації, потребує попередньої підготовки бази завдань	Середній/високий
AI-орієнтоване тестування	Використання алгоритмів штучного інтелекту для аналізу результатів та рекомендацій	Інтерактивність, індивідуальна траєкторія навчання, швидкий зворотний зв’язок	Високі вимоги до обчислювальних ресурсів та знань у розробці	Високий
Гібридні системи (Flutter + Java/Spring Boot)	Інтеграція адаптивних алгоритмів з сучасними фронтенд та бекенд технологіями	Кросплатформність, зручність користувача, масштабованість	Потребує технічної підтримки та оновлень	Високий

Порівняльний аналіз цих підходів показує, що найефективнішими є системи, які поєднують адаптивність із використанням сучасних технологій обробки даних. Вони дозволяють забезпечити високий рівень об’єктивності оцінювання, гнучкість використання та можливість інтеграції в існуючу інфраструктуру навчального закладу. Разом із цим, використання адаптивних комп’ютерних систем скорочує витрати часу викладачів на перевірку робіт та дає студентам зворотний зв’язок у режимі реального часу.

Отже, сьогодні розвиток автоматизованого тестування спрямований на створення систем, які одночасно відповідають трьом ключовим вимогам: точність оцінювання, адаптивність підхід до студентів та оптимальна інтеграція в освітній процес. Саме ці вимоги визначають актуальність досліджень у сфері адаптивних комп’ютерних систем оцінювання знань студентів.

1.3 Моделі адаптивного оцінювання

Адаптивне оцінювання знань студентів сьогодні є однією з найбільш ефективних методик вимірювання рівня підготовки, оскільки дозволяє враховувати індивідуальні особливості навчання кожного користувача. На відміну від класичних електронних тестів, де всі студенти отримують однаковий набір завдань, адаптивні системи динамічно формують тестові сесії на основі відповідей студента, що значно підвищує точність оцінювання та скорочує час проходження тесту.

Існує кілька основних моделей адаптивного оцінювання, які активно застосовуються у сучасних освітніх системах: моделі на основі правил, моделі на основі теорії ймовірнісного оцінювання знань (Item Response Theory, IRT), а також моделі з використанням алгоритмів машинного навчання та штучного інтелекту.

Модель на основі правил є найбільш простою та зрозумілою для реалізації. У таких системах заздалегідь визначаються рівні складності завдань та логіка переходу між ними. Наприклад, якщо студент правильно відповів на завдання середнього рівня складності, система пропонує завдання більш високого рівня, а у разі помилки — завдання простішого рівня. Основною перевагою цієї моделі є її прозорість та легкість налаштування, проте вона не враховує комплексні взаємозв'язки між знаннями та навичками студента. Тому точність оцінювання в умовах великих груп із різним рівнем підготовки може бути недостатньою.

Модель на основі теорії ймовірнісного оцінювання знань (IRT) дозволяє визначати рівень знань студента на основі математичних моделей, які оцінюють ймовірність правильної відповіді на конкретне завдання. У рамках IRT кожне питання характеризується певними параметрами: складністю, дискримінативністю та ймовірністю випадкової відповіді. Завдяки цим характеристикам система може більш точно оцінювати рівень знань та підбирати завдання, які максимально відповідають поточним компетенціям студента. IRT широко використовується у великих адаптивних системах, зокрема у міжнародних тестах і корпоративних платформах навчання. Недоліком моделі є складність налаштування параметрів і потреба у великій базі попередньо оцінених завдань.

Моделі на основі штучного інтелекту та машинного навчання є найсучаснішими і передбачають аналіз даних про поведінку студента під час проходження тестів, часу відповіді, історії результатів та інших метрик. На основі цих даних система може прогнозувати слабкі та сильні сторони студента, а також динамічно формувати індивідуальну освітню траєкторію. Подібні моделі дозволяють не тільки адаптувати складність завдань, а й давати рекомендації щодо додаткового навчання, надання пояснень чи інтерактивних матеріалів. Для реалізації таких моделей застосовуються алгоритми класифікації, регресії, глибокі нейронні мережі та методи рекомендацій.

У сучасних реалізаціях адаптивних систем важливим є поєднання моделей IRT та машинного навчання. Це забезпечує як математично обґрунтовану точність оцінювання, так і можливість постійного вдосконалення системи на основі накопичених даних. Завдяки використанню сучасних технологій веб-розробки та фреймворків, таких як Flutter/Dart для фронтенду та Java/Spring Boot для бекенду, ці системи стають кросплатформенними, швидкими та зручними для студентів і викладачів. Порівняння моделей адаптивного оцінювання наведено у таблиці 2.

Таблиця 2 — Порівняння моделей адаптивного оцінювання

Модель	**Принцип роботи**	**Переваги**	**Недоліки**	**Приклади застосування**
На основі правил	Підбір завдань за заздалегідь визначеними правилами складності	Проста реалізація, прозорість логіки, низькі витрати	Обмежена точність, не враховує складні взаємозв’язки знань	Невеликі навчальні системи, університетські тестові платформи
IRT (Item Response Theory)	Ймовірнісна модель оцінювання знань, завдання підбираються за параметрами складності, дискримінативності та ймовірності випадкової відповіді	Висока точність оцінювання, адаптація до рівня студента	Складне налаштування, потребує великої бази завдань	Великі адаптивні системи, міжнародні тести, корпоративні платформи
Машинне навчання та AI	Аналіз даних про студентів (результати, час відповіді, поведінка), формування індивідуальної траєкторії	Персоналізація навчання, рекомендації для повторення матеріалу, динамічна адаптація	Потребує значних обчислювальних ресурсів та накопиченої бази даних	Онлайн-платформи, інтерактивні освітні системи, системи з AI-рекомендаціями

Крім того, застосування адаптивних моделей дозволяє оптимізувати роботу викладачів, оскільки значна частина оцінювання відбувається

автоматично, а викладач отримує аналітичні дані про успішність студентів. Це дає змогу оперативно коригувати навчальний процес, зосереджувати увагу на проблемних темах та підвищувати загальну ефективність навчання.

Таким чином, моделі адаптивного оцінювання забезпечують високу точність оцінювання знань студентів, персоналізацію освітнього процесу та ефективну інтеграцію сучасних технологій обробки даних, що робить їх актуальними для впровадження у вищій освіті та інтерактивних онлайн-платформах.

1.3.1 Теорія Item Response Theory (IRT)

Item Response Theory (IRT) — це математична модель оцінювання знань, яка використовується для адаптивного тестування. Основна ідея IRT полягає в тому, що ймовірність правильної відповіді студента на конкретне тестове завдання залежить від двох факторів: рівня знань студента та характеристик самого завдання. Кожне завдання описується параметрами складності, дискримінації та ймовірності випадкової відповіді.

Параметр складності визначає, наскільки завдання є складним для середнього студента. Параметр дискримінації показує, наскільки завдання здатне розрізняти студентів із різним рівнем знань, тобто чим вище дискримінація, тим точніше завдання визначає знання студента. Параметр ймовірності випадкової відповіді враховує ймовірність того, що студент відповість правильно випадково, наприклад при завданні з кількома варіантами відповідей.

Системи на основі IRT дозволяють створювати адаптивні тести, які змінюють складність завдань у реальному часі. Студент отримує питання, що відповідають його рівню знань, а система оновлює оцінку після кожної відповіді. Це забезпечує точне визначення рівня знань при меншій кількості питань, ніж у традиційних тестах.

1.3.2 Алгоритми підбору складності тестових завдань

Алгоритми підбору складності завдань є ключовим елементом адаптивних систем оцінювання. Найпоширеніший підхід ґрунтується на принципі «золотої середини»: якщо студент правильно відповів на попереднє завдання, наступне питання буде складнішим; якщо відповідь була помилковою — наступне завдання буде простішим.

Сучасні системи використовують більш точні алгоритми, що спираються на IRT. Кожне питання має оцінку складності, і алгоритм підбирає таке, яке максимально збільшує інформаційне значення тесту для конкретного студента. Додатково застосовуються алгоритми на основі машинного навчання, які враховують історію відповідей студента, час на виконання завдань, а також типи помилок. Це дозволяє формувати індивідуальну траєкторію навчання та коригувати оцінку знань у режимі реального часу.

Крім того, деякі алгоритми враховують баланс між складністю та мотивацією студента. Наприклад, занадто багато складних питань поспіль може знизити увагу та ефективність проходження тесту, тому система автоматично підбирає чергування легких та складних завдань.

1.3.3 Переваги та недоліки адаптивних систем

Адаптивні системи оцінювання знань мають кілька ключових переваг. По-перше, вони дозволяють отримати більш точну оцінку рівня підготовки студента за меншу кількість завдань, що економить час. По-друге, такі системи забезпечують індивідуальний підхід, враховуючи сильні та слабкі сторони кожного студента. По-третє, завдяки використанню алгоритмів та IRT зменшується людський фактор та підвищується об’єктивність оцінювання.

Недоліки таких систем пов’язані з технічною складністю та потребою у великій базі якісно анотованих завдань. Для правильної роботи системи необхідно налаштувати алгоритми підбору складності, провести тестування та валідацію моделей. Крім того, адаптивні системи потребують обчислювальних

ресурсів для обробки даних у реальному часі та збереження історії проходження тестів.

Таким чином, адаптивні системи оцінювання знань є ефективним інструментом для сучасної освіти, що дозволяє поєднувати точність, персоналізацію та швидкість оцінювання, але потребує серйозної технічної підтримки та планування бази завдань.

1.4 Аналіз сучасних систем тестування

У сучасній вищій освіті використання електронних систем тестування стало стандартом для оцінювання знань студентів. Ці платформи дозволяють автоматизувати перевірку, підвищити об’єктивність оцінювання та надати студентам швидкий зворотний зв’язок. Водночас, рівень адаптивності різних систем значно відрізняється, що визначає ефективність оцінювання для студентів із різним рівнем підготовки.

Сучасні системи тестування можна класифікувати на три основні групи: відкриті платформи з адаптивними модулями, комерційні рішення та інші альтернативні освітні платформи. Кожна з цих груп має власні особливості щодо гнучкості, вартості впровадження, наявності аналітичних інструментів та можливостей інтеграції з навчальними ресурсами.

Нижче наведено приклад порівняльної таблиці ключових характеристик сучасних систем тестування (Табл.3).

Таблиця 3 – Порівняння сучасних систем тестування

Система	**Тип**	**Адаптивність**	**Інтеграція**	**Вартість**	**Підтримка аналітики**
Moodle	Відкрита	Модулі адаптивного тестування	LMS, сторонні плагіни	Безкоштовна, оплата хостингу	Статистика, звіти
Blackboard	Комерційна	Обмежена адаптивність	LMS, SCORM	Висока	Розширена аналітика, дашборди
Smart Sparrow	Комерційна	Повністю адаптивна	API, інтеграція з LMS	Середня	Індивідуальні траєкторії, рекомендації
Kahoot!	Альтернативна	Мінімальна	Веб, мобільні додатки	Безкоштовна/преміум	Основна статистика
ClassMarker	Комерційна	Частково адаптивна	API, LMS	Середня	Звіти, аналітика
Canvas	Відкрита/комерційна	Модулі адаптивного тестування	LMS, сторонні інтеграції	Середня	Статистика, дашборди

Таким чином, вибір платформи залежить від цілей навчального закладу, потреб студентів та бюджету на впровадження.

1.4.1 Системи Moodle та їх адаптивні модулі

Moodle є однією з найбільш популярних систем управління навчанням (LMS) із відкритим кодом. Платформа підтримує створення тестів та опитувань із використанням модулів адаптивного тестування, таких як «Adaptive Quiz» або плагіни для реалізації Item Response Theory. Ці модулі дозволяють динамічно змінювати складність завдань залежно від попередніх відповідей студента.

Серед переваг Moodle — низька вартість впровадження, велика спільнота розробників, наявність інтеграції з іншими освітніми ресурсами та можливість ведення детальної статистики результатів. Недоліки включають необхідність технічної підтримки, налаштування серверної інфраструктури та обмежену адаптивність у порівнянні з комерційними платформами.

1.4.2 Комерційні адаптивні платформи

Комерційні платформи, такі як Smart Sparrow або Blackboard, пропонують готові рішення для адаптивного навчання та оцінювання. Вони забезпечують більш гнучкі алгоритми підбору завдань, індивідуальні освітні траєкторії та аналітичні інструменти для викладачів.

Переваги таких систем включають стабільну підтримку, можливість масштабування та інтеграцію з існуючими LMS. Недоліки — висока вартість ліцензій, обмеження у налаштуванні та залежність від постачальника. Комерційні платформи зазвичай підходять для великих університетів або організацій, які потребують високої надійності та розширеної аналітики.

1.4.3 Інші рішення на ринку освітніх систем

Інші рішення, наприклад Kahoot! або ClassMarker, пропонують прості інструменти для тестування, які не потребують складної інфраструктури. Такі платформи часто використовуються для коротких опитувань, повторення матеріалу та інтерактивних вправ.

Переваги включають легкість використання, швидкий старт та мінімальні витрати. Недоліки — обмежена адаптивність, базова аналітика та неможливість глибокої інтеграції з LMS. Такі системи часто застосовуються у змішаному навчанні або для самостійного повторення студентами.

1.5 Висновки з розділу 1

Було проведено детальний аналіз проблеми оцінювання знань студентів у сучасній вищій освіті та досліджено існуючі підходи до автоматизованого й адаптивного тестування. Встановлено, що традиційні електронні тести з фіксованим набором завдань відзначаються простотою реалізації, однак не враховують індивідуальні особливості студентів і не завжди дозволяють об’єктивно оцінити реальний рівень знань у групах із різним ступенем підготовки.

Проаналізовано моделі адаптивного оцінювання, зокрема застосування теорії Item Response Theory (IRT), алгоритмів динамічного підбору складності тестових завдань, а також використання методів штучного інтелекту для формування індивідуальних освітніх траєкторій. Адаптивні системи дають змогу зменшити кількість тестових питань без втрати точності оцінювання, підвищити об’єктивність результатів та надавати персоналізовані рекомендації щодо подальшого навчання, що позитивно впливає на мотивацію студентів і ефективність засвоєння матеріалу.

У межах аналізу сучасних систем тестування було виділено три основні категорії: відкриті платформи з адаптивними модулями (наприклад, Moodle), комерційні адаптивні платформи (Smart Sparrow, Blackboard) та інші освітні рішення, орієнтовані на інтерактивність і швидке оцінювання знань (Kahoot!, ClassMarker). Проведене порівняння показало, що відкриті платформи забезпечують гнучкість налаштування та доступність, комерційні системи пропонують розширені аналітичні інструменти й стабільну технічну підтримку, тоді як легкі альтернативні платформи доцільні для оперативного опитування та залучення студентів до навчального процесу.

Проведений аналіз свідчить, що найбільш ефективними є системи, які поєднують адаптивний підхід із використанням сучасних технологій обробки та аналізу даних. Застосування адаптивних комп’ютерних систем оцінювання знань забезпечує високу точність оцінювання завдяки динамічному підбору завдань відповідно до рівня підготовки студента, персоналізацію навчального процесу через формування індивідуальної освітньої траєкторії, оптимізацію роботи викладача шляхом скорочення часу на перевірку результатів, а також гнучку інтеграцію в існуючу інфраструктуру навчального закладу з використанням сучасних технологій фронтенду та бекенду, зокрема Flutter/Dart і Java/Spring Boot.

Отримані результати підтверджують актуальність розробки адаптивної комп’ютерної системи оцінювання знань студентів, яка враховує індивідуальні особливості користувачів, базується на сучасних технологічних рішеннях і

спрямована на підвищення ефективності навчального процесу. Такий аналіз формує теоретичне підґрунтя для подальшого проєктування та практичної реалізації системи.

РОЗДІЛ 2 АНАЛІЗ ЗАСОБІВ ТА ТЕХНОЛОГІЙ ДЛЯ РОЗРОБКИ АДАПТИВНОЇ СИСТЕМИ

2.1 Архітектурні підходи до побудови веб- та мобільних систем

Сучасні веб- та мобільні системи оцінювання знань студентів потребують ефективної архітектури, яка забезпечує масштабованість, надійність та швидку взаємодію користувача із системою. Основні архітектурні підходи можна умовно поділити на три категорії:

Монолітна архітектура — вся функціональність системи реалізована у вигляді єдиного додатка. Переваги: простота розгортання та тестування. Недоліки: складність масштабування та внесення змін у великих проектах.

Мікросервісна архітектура — система розбивається на незалежні сервіси, кожен з яких відповідає за конкретну функцію (керування тестами, авторизація, аналіз результатів). Переваги: масштабованість, гнучкість, легкість інтеграції нових модулів. Недоліки: підвищена складність розгортання та моніторингу.

Клієнт-серверна архітектура — фронтенд взаємодіє із сервером через API. Класичним прикладом є поєднання мобільного додатку на Flutter із бекендом на Java/Spring Boot. Така архітектура дозволяє підтримувати одночасно веб- і мобільний клієнт.

Нижче наведено порівняльну таблицю основних архітектурних підходів для освітніх платформ (Таблиця 4).

Таблиця 4 – Основні архітектурні підходи для освітніх платформ

Архітектура	**Переваги**	**Недоліки**	**Приклади використання**
Монолітна	Простота реалізації, тестування	Складність масштабування	Moodle, ClassMarker
Мікросервісна	Масштабованість, гнучкість	Складність розгортання	Blackboard, Coursera
Клієнт-серверна	Підтримка веб і мобільних клієнтів	Необхідність налаштування API	Smart Sparrow, адаптивні системи

Таким чином, для адаптивної системи оцінювання знань студентів оптимальним є поєднання клієнт-серверної архітектури з окремими мікросервісами для обробки аналітики та рекомендацій.

2.2 Інструменти для розробки фронтенд-частини

Фронтенд-частина адаптивної системи оцінювання знань студентів відповідає за взаємодію користувача із системою. Вона включає інтерфейс для проходження тестів, перегляду результатів, отримання рекомендацій та аналітичних звітів. Сучасні підходи до розробки фронтенду передбачають використання кросплатформових фреймворків, що дозволяють створювати одночасно веб- та мобільні версії системи.

Одним із найбільш популярних рішень для розробки мобільних і веб-застосунків є Flutter — відкритий фреймворк від компанії Google. Він дозволяє створювати нативні інтерфейси користувача для Android, iOS та веб із використанням єдиного коду. Flutter працює на основі рендерингу власного движка, що забезпечує високу продуктивність та точне відтворення дизайну на різних пристроях. Для розробки адаптивної системи оцінювання знань Flutter дозволяє реалізувати інтерактивні тести, динамічне відображення результатів та анімації, що підвищують залученість користувачів.

Мовою програмування, яка використовується у Flutter, є Dart. Dart підтримує об’єктно-орієнтоване програмування та забезпечує ефективну роботу з асинхронними операціями, що важливо для системи, яка взаємодіє з сервером у реальному часі. Основні переваги Dart полягають у простоті синтаксису, можливості гарячого перезапуску (hot reload) під час розробки та високій продуктивності на мобільних пристроях. Використання Dart дозволяє реалізувати логіку валідації відповідей студентів, підбір адаптивних завдань і динамічне відображення результатів у зручному інтерфейсі.

Для розробки фронтенду також важливими є бібліотеки та плагіни Flutter, які спрощують роботу з базами даних, графікою, мережевими запитами

та анімаціями. У системі адаптивного оцінювання можна використовувати плагіни для побудови графіків успішності студентів, інтеграції push-повідомлень, а також для роботи з камерою чи мікрофоном у випадку інтерактивних завдань.

Таким чином, комбінація Flutter + Dart дозволяє створювати високопродуктивну фронтенд-частину адаптивної системи оцінювання знань студентів із гнучкою адаптивною логікою, інтерактивними тестами та інтуїтивним інтерфейсом для користувачів будь-якого віку.

2.2.1 Фреймворк Flutter

Flutter — це відкритий фреймворк для кросплатформової розробки, який використовує власний графічний движок Skia для рендерингу інтерфейсу. Завдяки цьому, дизайн відображається однаково на всіх платформах, незалежно від версії ОС чи типу пристрою. Flutter дозволяє будувати складні анімації, інтерактивні графічні елементи та динамічні списки, що особливо важливо для адаптивного тестування, де студент бачить різні завдання залежно від своїх відповідей.

Ключовою особливістю Flutter є Widget-based architecture. Кожен елемент інтерфейсу — це віджет, що можна комбінувати та наслідувати для створення нових компонентів. Для адаптивної системи оцінювання знань це дозволяє реалізувати кастомні тести, таймери, прогрес-бари, відображення статистики та рекомендацій у режимі реального часу.

Flutter також підтримує роботу з мережевими API через бібліотеки http та Dio, що дозволяє обмінюватися даними з бекендом Java/Spring Boot. Завдяки асинхронним запитам дані тестів можуть оновлюватися без перезавантаження інтерфейсу, забезпечуючи плавну взаємодію користувача із системою.

2.2.2 Мова Dart

Dart — це мова програмування, яка розроблена спеціально для Flutter. Вона поєднує високу продуктивність із простим синтаксисом, що дозволяє

швидко реалізовувати складні логічні блоки системи. Основними особливостями Dart є:

1. Об’єктно-орієнтованість, що дозволяє моделювати користувацькі об’єкти, завдання, результати та оцінки.
2. Асинхронність за допомогою Future та Stream, що забезпечує обробку запитів до серверу без блокування інтерфейсу.
3. Гарячий перезапуск (hot reload), що значно прискорює розробку та тестування адаптивної логіки.

У адаптивній системі оцінювання Dart використовується для:

- формування динамічного набору тестових завдань;
- розрахунку балів та визначення складності наступних питань;
- відображення персоналізованих рекомендацій;
- інтеграції з локальним кешем та базою даних для збереження прогресу студента.

2.3 Інструменти для бекенд-розробки

Бекенд-частина адаптивної системи оцінювання знань студентів відповідає за обробку запитів користувачів, зберігання даних, формування адаптивного тесту та забезпечення безпеки системи. Для розробки бекенду сучасні освітні платформи часто використовують мови програмування та фреймворки, які підтримують масштабованість, безпеку та ефективну інтеграцію з фронтендом.

Одним із найбільш популярних рішень є Java — мова з багаторічним досвідом застосування у корпоративних та освітніх проектах. Java забезпечує стабільність роботи, багатопоточність, підтримку великої кількості бібліотек і фреймворків для роботи з базами даних, веб-сервісами та безпекою. Для адаптивної системи оцінювання знань Java дозволяє ефективно обробляти численні запити студентів одночасно, формувати персоналізовані тести та забезпечувати логіку підбору складності завдань.

Фреймворк Spring Boot значно спрощує створення бекенд-застосунків на Java. Він надає готові механізми для організації REST API, інтеграції з базами даних, управління безпекою та обробки помилок. Використання Spring Boot дозволяє швидко реалізувати адаптивну логіку системи, наприклад:

- динамічне формування набору питань для кожного студента;
- розрахунок балів та рекомендацій;
- збереження результатів у базі даних;
- інтеграцію з фронтендом через REST API.

2.3.1 Мова Java

Java є об’єктно-орієнтованою мовою програмування високого рівня. Основні переваги Java для розробки бекенду адаптивної системи оцінювання знань:

1. Масштабованість — можливість обслуговувати великий потік запитів студентів одночасно.
2. Портативність — JVM дозволяє запускати код на різних платформах без модифікацій.
3. Багатопоточність — підтримка одночасного виконання численних запитів.
4. Безпека — широкий спектр бібліотек для автентифікації та авторизації користувачів.

Java також підтримує інтеграцію з сучасними базами даних, кешуванням, чергами повідомлень і веб-сервісами, що забезпечує надійну роботу адаптивної системи у реальному часі.

2.3.2 Фреймворк Spring Boot

Spring Boot дозволяє швидко створювати веб-додатки без складної конфігурації. Для системи адаптивного оцінювання знань він використовується для:

- організації REST API для взаємодії з фронтендом;

- налаштування безпеки користувачів (Spring Security);
- автоматичної роботи з базами даних через JPA/Hibernate;
- логування та моніторингу запитів.

Spring Boot забезпечує стандартизовану архітектуру, що спрощує подальшу підтримку та масштабування системи.

2.3.3 Реалізація REST API

REST API є проміжною ланкою між фронтендом і бекендом. Вона дозволяє фронтенду отримувати дані про студентів, тестові завдання, результати та рекомендації у форматі JSON. REST API реалізується за допомогою контролерів Spring Boot, які обробляють HTTP-запити типу GET, POST, PUT та DELETE.

Основні функції REST API в адаптивній системі:

- надсилання тестових завдань студенту;
- отримання відповідей та оцінок;
- динамічне формування рекомендацій;
- управління користувачами та ролями.

REST API забезпечує незалежність фронтенду та бекенду, що спрощує оновлення інтерфейсу та логіки системи без порушення роботи.

2.4 Засоби зберігання та обробки даних

У системі адаптивного оцінювання знань студентів використовується значна кількість даних: бази запитань, відповіді студентів, статистика, аналітика результатів та рекомендації. Основним підходом є використання реляційних баз даних, які дозволяють організувати структуру даних у вигляді таблиць із зв’язками між ними.

Реляційні бази даних підтримують SQL-запити, транзакції та механізми цілісності даних, що критично важливо для системи, де точність результатів безпосередньо впливає на оцінки студентів.

2.4.1 Реляційні бази даних

У системі використовується реляційна БД (наприклад, PostgreSQL або MySQL). Таблиця 5 ілюструє основні сутності та їх зв’язки.

Таблиця 5 – Основні сутності та їх зв’язки

Сутність	**Атрибути**	**Зв’язки**
Student	id, name, email, group	1:N → Responses
Test	id, title, description, difficulty_level	1:N → Questions
Question	id, text, options, correct_answer, score	N:1 → Test, 1:N → Responses
Response	id, student_id, question_id, answer, score	N:1 → Student, N:1 → Question
Recommendation	id, student_id, content, date_created	N:1 → Student

Ця структура дозволяє ефективно зберігати всі дані, забезпечувати швидкий пошук, генерацію адаптивного тесту та формування персоналізованих рекомендацій.

2.5 Аналіз протоколів взаємодії клієнт-сервер

Для інтеграції фронтенду і бекенду використовуються різні протоколи взаємодії. Найпоширеніший протокол у сучасних веб- і мобільних системах — HTTP/HTTPS, який забезпечує передачу даних через REST API у форматі JSON.

Протокол дозволяє ефективно обробляти запити та відповіді, працювати з кешуванням та забезпечує безпеку даних через HTTPS. Альтернативою може бути WebSocket, який забезпечує двосторонню комунікацію в реальному часі, що підтверджується порівняльними характеристиками основних протоколів комунікації, наведеними в таблиці 6.

Таблиця 6 – Основні протоколи комунікації

Протокол	**Тип комунікації**	**Переваги**	**Недоліки**
HTTP/HTTPS	Запит-відповідь	Простота, сумісність, безпека	Повільніше для реального часу
WebSocket	Двостороння	Миттєвий обмін даними	Складніша реалізація, навантаження
gRPC	Виклики функцій	Висока продуктивність, типізація	Потребує додаткових бібліотек

В адаптивній системі оцінювання HTTP/HTTPS через REST API використовується для стандартних запитів (отримання завдань, надсилання відповідей), а WebSocket може застосовуватися для відображення результатів і прогресу студентів у реальному часі.

2.6 Висновки з розділу 2

Було розглянуто архітектурні підходи та інструменти для розробки веб- і мобільної частин адаптивної системи оцінювання знань студентів. Ключовими компонентами такої системи є фронтенд, реалізований на базі Flutter/Dart, та бекенд, побудований з використанням Java і Spring Boot із підтримкою REST API та реляційної бази даних.

Фронтенд-частина забезпечує інтуїтивно зрозумілий інтерфейс користувача, підтримку інтерактивного проходження тестів і відображення персоналізованих рекомендацій. Бекенд відповідає за обробку запитів, реалізацію бізнес-логіки адаптивного тестування, управління даними та забезпечення безпеки системи. Використання реляційної бази даних дозволяє організувати структуроване зберігання інформації, забезпечити швидкий доступ до результатів тестування та підтримати інтеграцію з аналітичними модулями.

Аналіз протоколів клієнт-серверної взаємодії показав доцільність використання REST API для основного обміну даними в поєднанні з

WebSocket для сценаріїв, що потребують оновлення інформації в реальному часі. Такий підхід дозволяє досягти оптимального балансу між продуктивністю, масштабованістю та зручністю взаємодії користувачів із системою.

Поєднання зазначених технологій створює передумови для побудови масштабованої, безпечної та ефективної адаптивної комп’ютерної системи оцінювання знань студентів, здатної відповідати сучасним вимогам освітніх закладів і забезпечувати комфортну роботу як для студентів, так і для викладачів.

РОЗДІЛ 3 РОЗРОБКА АДАПТИВНОЇ КОМП'ЮТЕРНОЇ СИСТЕМИ ОЦІНЮВАННЯ ЗНАНЬ СТУДЕНТІВ

3.1 Проєктування системи

Проєктування програмної системи адаптивного тестування є ключовим етапом, що визначає подальшу якість, надійність і розширюваність усієї розробки. У цьому розділі виконується формалізація вимог, побудова моделі предметної області, а також створення архітектури, яка має поєднати складну логіку адаптивного підбору тестових завдань, модулі аналітики, механізми оцінювання та взаємодію користувача з системою. Адаптивне тестування накладає підвищені вимоги до структурованості, оскільки система має реагувати на поведінку користувача в режимі реального часу, обираючи наступне запитання не випадково, а на основі розрахованих параметрів.

3.1.1 Моделювання вимог

Моделювання вимог є фундаментальною фазою життєвого циклу розробки, адже саме на цьому етапі визначаються функціональні та нефункціональні характеристики системи. Для адаптивної системи тестування вимоги формуються у трьох вимірах: користувацькому, функціональному, технологічному.

Користувацькі вимоги

Адаптивна система має обслуговувати три основні типи користувачів:

Студент (респондент):

- проходить тест у мобільному застосунку;
- отримує запитання, складність яких змінюється автоматично;
- повинен бачити прогрес, проміжні повідомлення, завершальний результат;
- очікує інтуїтивний інтерфейс без необхідності попередньої інструкції.

Викладач / адміністратор тестів:

- створює та редагує банк завдань;
- вказує параметри IRT (або дає системі можливість оцінити їх автоматично);
- керує сесіями тестувань та групами студентів;
- отримує детальні статистичні звіти про результати.

Система (автоматизований модуль адаптації):

- аналізує відповіді студента;
- коригує складність наступного питання;
- припиняє тест після досягнення заданої точності оцінки;
- забезпечує узгодженість усіх параметрів.

Функціональні вимоги:

Основний функціонал системи включає такі групи:

Управління тестами та питаннями:

- створення тестів з довільною кількістю питань;
- можливість класифікації запитань за темами, рівнем складності, типом;
- зберігання параметрів моделей IRT: складність, дискримінація, імовірність вгадування;
- імпорт/експорт банку питань.

Адаптивний механізм:

- визначення початкової складності тесту;
- алгоритм підбору наступного питання на основі оцінки рівня студента;
- збереження проміжних значень параметра θ (theta) після кожної відповіді;
- завершення тесту за умовами стабільності оцінки.

Процедура тестування на клієнтській частині:

- авторизація студента;

- відправлення відповіді на сервер;
- отримання наступного питання відповідно до адаптивного алгоритму;
- коректна обробка мережевих переривань.

Аналітика та звітність:

- візуалізація індивідуальної оцінки рівня студента;
- статистика складності питань;
- інтегральні показники успішності групи;
- аналіз валідності тесту.

Нефункціональні вимоги:

1. Продуктивність: система повинна відповідати менш ніж за 200 мс на кожен API-запит, щоб користувач не відчував затримок між питаннями.
2. Масштабованість: бекенд має підтримувати горизонтальне масштабування за рахунок контейнеризації.
3. Надійність: дані про відповіді не повинні втрачатися навіть при розриві з’єднання.
4. Безпека: автентифікація користувачів, шифрування паролів, захист від зовнішніх атак.
5. Мобільність: фронтенд має працювати однаково стабільно на Android і iOS.

3.1.2 Модель предметної області

Модель предметної області визначає всі сутності, які присутні в системі адаптивного тестування, а також їхні атрибути і зв’язки між ними. Таке моделювання дозволяє систематизувати логіку роботи, а надалі — побудувати структуру бази даних і API.

Основні сутності

1. User (Користувач).

Атрибути:

- id;

- ім’я;
- email;
- роль (student, admin);
- хешований пароль.

Зв’язки:

- User проходить багато тестових сесій.

2. Test (Тест).

Атрибути:

- id;
- назва;
- опис;
- автор;
- дата створення.

Зв’язки:

- містить багато питань (Question);
- має багато сесій проходження (TestSession).

3. Question (Питання).

Атрибути:

- id;
- текст запитання;
- варіанти відповіді;
- правильний варіант;
- параметри IRT (a, b, c);
- рівень складності (як допоміжний атрибут).

Зв’язки:

- належить одному тесту;
- використовується в багатьох сесіях.

4. TestSession (Сесія тестування).

Атрибути:

- id;
- id користувача;
- id тесту;
- поточний параметр θ;
- статус (active, finished);
- час початку/завершення.

Зв’язки:

- містить багато відповідей Answer.

5. Answer (Відповідь).

Атрибути:

- id;
- id питання;
- id сесії;
- отримана відповідь;
- чи правильна;
- оновлений параметр θ.

Логіка зв’язків

Між сутностями існують такі типи зв'язків:

- User — TestSession: один до багатьох; один користувач може пройти багато тестів.
- Test — Question: один до багатьох; тест містить довільну кількість завдань.
- TestSession — Answer: один до багатьох; кожна відповідь входить у конкретну сесію.

Ця модель дозволяє зберігати історію всіх проходжень, а також забезпечує можливість повторного аналізу даних для корекції параметрів IRT.

3.1.3 Архітектурна схема системи

Архітектура адаптивної системи тестування має бути модульною, структурованою та здатною обробляти велике навантаження. У рамках даної роботи застосовується клієнт–серверна архітектура, де серверна частина забезпечує логіку тестування, а клієнтська — інтерфейс користувача.

1. Рівень бекенду

1.1. Бекенд реалізовано на Spring Boot, що забезпечує:

- модульність;
- розподіл відповідальностей;
- легку інтеграцію з базами даних;
- високу продуктивність.

1.2. Основні модулі:

• API-модуль

- приймає запити від мобільного додатку;
- повертає питання;
- приймає відповіді та обчислює нове значення θ.

• Модуль адаптивного алгоритму

Реалізує:

- модель IRT;
- логіку оновлення θ;
- підбір наступного питання згідно з інформаційною функцією.

• Модуль безпеки

Використовує Spring Security:

- автентифікація;
- авторизація;
- JWT-токени;
- захист API.

• Модуль роботи з базою даних

На основі JPA/Hibernate:

- сутності;
- репозиторії;
- транзакційність.

2. Рівень фронтенду

Фронтенд створюється у Flutter, що дозволяє розробляти застосунок одночасно для Android та iOS.

2.1. Основні модулі:

- UI шар — екрани, віджети, модальні вікна;
- Service шар — робота з API;
- State management — зміна стану тестування в реальному часі;
- Offline-layer — кешування запитів.

3. Рівень бази даних

- Використовується реляційна БД (PostgreSQL).

Структура таблиць відповідає моделі предметної області.

Додатково:

- зберігання логів;
- індексація по найбільш використовуваних полях;
- можливість горизонтального масштабування.

4. Інтеграційний рівень

Забезпечує:

- передачу даних між Flutter-додатком і сервером;
- консистентність сесій тестування;
- коректну обробку помилок.

3.2 Реалізація бекенд-частини

Реалізація серверної частини адаптивної системи тестування є ключовим етапом, оскільки саме бекенд відповідає за обробку запитів, логіку адаптивного підбору запитань, збереження даних, аналітику результатів та

забезпечення безпеки користувачів. У цьому розділі розглядається налаштування середовища Spring Boot, архітектурні рішення REST API, механізми адаптації, модуль автентифікації та авторизації, а також структура бази даних та моделі сутностей. Бекенд є центральною частиною системи, яка забезпечує взаємодію між користувачем та інтелектуальним механізмом оцінювання.

3.2.1 Налаштування середовища Spring Boot

Spring Boot було обрано як основу бекенд-архітектури завдяки його гнучкості, широкій екосистемі модулів, простоті масштабування та чіткій структурі, яка забезпечує високу продуктивність. Налаштування середовища розпочалося зі створення базового проєкту, який включає необхідні залежності та попередньо визначену архітектуру.

1. Ініціалізація проєкту

Для створення проєкту використовується Spring Initializr, де підключаються такі модулі:

- Spring Web — для реалізації REST API;
- Spring Data JPA — для роботи з базою даних через ORM;
- PostgreSQL Driver — як драйвер реляційної БД;
- Spring Security — для автентифікації та авторизації;
- Validation — для валідації вхідних запитів;
- Lombok — для скорочення кількості шаблонного коду.

Лістинг 1 —pom.xml з залежностями

```
<?xml version="1.0" encoding="UTF-8"?>
<project xmlns="http://maven.apache.org/POM/4.0.0"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
         xsi:schemaLocation="http://maven.apache.org/POM/4
.0.0 https://maven.apache.org/xsd/maven-4.0.0.xsd">
         <modelVersion>4.0.0</modelVersion>
         <parent>
```

```
            <groupId>org.springframework.boot</groupId>
            <artifactId>spring-boot-starter-
parent</artifactId>
            <version>4.0.0</version>
            <relativePath/>  <!--  lookup  parent  from
repository -->
        </parent>
        <groupId>com.adaptive_test</groupId>
        <artifactId>test</artifactId>
        <version>0.0.1-SNAPSHOT</version>
        <name>test</name>
        <description>Adaptive computer system for student
knowledge assessment</description>
        <url/>
        <licenses>
            <license/>
        </licenses>
        <developers>
            <developer/>
        </developers>
        <scm>
            <connection/>
            <developerConnection/>
            <tag/>
            <url/>
        </scm>
        <properties>
            <java.version>17</java.version>
            <jjwt.version>0.11.5</jjwt.version>
        </properties>
        <dependencies>
            <!-- Web / REST -->
            <dependency>
```

```
    <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
web</artifactId>
            </dependency>

            <!-- JPA / Hibernate -->
            <dependency>

    <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-data-
jpa</artifactId>
            </dependency>

            <!-- Security -->
            <dependency>

    <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
security</artifactId>
            </dependency>

            <!-- Validation (для DTO) -->
            <dependency>

    <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
validation</artifactId>
            </dependency>

            <!-- JWT -->
            <dependency>
                <groupId>io.jsonwebtoken</groupId>
```

```
                <artifactId>jjwt-api</artifactId>
                <version>${jjwt.version}</version>
            </dependency>
            <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt-impl</artifactId>
                <version>${jjwt.version}</version>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>io.jsonwebtoken</groupId>
                <artifactId>jjwt-jackson</artifactId>
                <version>${jjwt.version}</version>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>org.postgresql</groupId>
                <artifactId>postgresql</artifactId>
                <scope>runtime</scope>
            </dependency>
            <dependency>
                <groupId>com.h2database</groupId>
                <artifactId>h2</artifactId>
                <scope>runtime</scope>
            </dependency>
            <dependency>

    <groupId>org.springframework.boot</groupId>
                <artifactId>spring-boot-starter-
test</artifactId>
                <scope>test</scope>
            </dependency>
        </dependencies>
```

```
        <build>
            <plugins>
                <plugin>

    <groupId>org.springframework.boot</groupId>
                    <artifactId>spring-boot-maven-
plugin</artifactId>
                </plugin>
            </plugins>
        </build>

    </project>
```

2. Налаштування конфігурації

Файл application.propertiesмістить налаштування для:

- підключення до бази даних;
- портів сервера;
- параметрів безпеки;
- механізмів логування;
- крос-доменного доступу (CORS).

Лістинг 2 — application.properties– налаштування БД, серверу, JPA

```
    spring.application.name=adaptive-testing

    spring.datasource.url=jdbc:h2:mem:adaptive_testing;DB_C
LOSE_DELAY=-1;MODE=PostgreSQL
    spring.datasource.username=qryxq
    spring.datasource.password=qwerty
    spring.datasource.driver-class-name=org.h2.Driver

    spring.jpa.hibernate.ddl-auto=update
```

```
spring.jpa.show-sql=true
spring.jpa.properties.hibernate.format_sql=true

spring.jpa.properties.hibernate.jdbc.time_zone=UTC
spring.jpa.database-
platform=org.hibernate.dialect.H2Dialect

spring.h2.console.enabled=true
spring.h2.console.path=/h2-console

app.security.jwt.secret=change-this-secret-in-prod
app.security.jwt.expiration-ms=3600000

server.error.include-message=always

server.port=8081
```

3. Структура проєкту

Проєкт структуровано за принципами DDD (Domain-Driven Design):

- controller — обробка HTTP-запитів;
- service — бізнес-логіка;
- repository — доступ до бази даних;
- model/entity — сутності;
- security — конфігурації безпеки;
- dto — об’єкти передачі даних;
- mapper — перетворення моделей у DTO та навпаки.

Таке розділення забезпечує масштабованість і легкість підтримки.

3.2.2 Реалізація REST API для тестування

REST API забезпечує взаємодію між мобільним застосунком і сервером у форматі JSON. API спроєктовано так, щоб:

- забезпечити чіткий контракт між клієнтом і сервером;

- гарантувати стабільність відповідей;
- легко доповнювати новий функціонал.

1. Основні ендпоінти системи

REST API поділений на кілька логічних модулів:

1) /auth

- POST /register
- POST /login

2) /tests

- GET /tests — отримання списку доступних тестів;
- GET /tests/{id} — отримання даних тесту;

3) /session

- POST /start — почати нову сесію тестування;
- POST /answer — надіслати відповідь;
- GET /status — отримати поточну інформацію про сесію;
- POST /finish — примусово завершити тест.

4) /questions

- GET /next — отримання наступного адаптивного запитання.

Лістинг 3 — контролер TestController

```
package com.adaptive.test.controller;

import com.adaptive.test.domain.TestEntity;
import com.adaptive.test.dto.test.TestDto;
import com.adaptive.test.service.TestService;
import jakarta.validation.Valid;
import org.springframework.http.ResponseEntity;
import
org.springframework.security.access.prepost.PreAuthorize;
import
org.springframework.web.bind.annotation.GetMapping;
import
org.springframework.web.bind.annotation.PostMapping;
import
org.springframework.web.bind.annotation.RequestBody;
import
org.springframework.web.bind.annotation.RequestMapping;
import
org.springframework.web.bind.annotation.RestController;
```

```
    import java.util.List;
    import java.util.stream.Collectors;

    @RestController
    @RequestMapping("/tests")
    public class TestController {

        private final TestService testService;

        public TestController(TestService testService) {
            this.testService = testService;
        }

        @GetMapping
        @PreAuthorize("hasAnyRole('STUDENT','TEACHER')")
        public ResponseEntity<List<TestDto>> list() {
            List<TestDto> tests =
testService.findAll().stream()
                    .map(this::toDto)
                    .collect(Collectors.toList());
            return ResponseEntity.ok(tests);
        }

        @PostMapping
        @PreAuthorize("hasRole('TEACHER')")
        public ResponseEntity<TestDto> create(@Valid
@RequestBody TestDto dto) {
            TestEntity entity = new TestEntity();
            entity.setTitle(dto.getTitle());
            entity.setDescription(dto.getDescription());

entity.setBaseDifficulty(dto.getBaseDifficulty());

            TestEntity saved = testService.create(entity);
            return ResponseEntity.ok(toDto(saved));
        }

        private TestDto toDto(TestEntity entity) {
            TestDto dto = new TestDto();
            dto.setId(entity.getId());
            dto.setTitle(entity.getTitle());
            dto.setDescription(entity.getDescription());

dto.setBaseDifficulty(entity.getBaseDifficulty());
            return dto;
        }
    }
```

2. Обмін даними між клієнтом і сервером

Усі дані передаються через DTO-класи. Наприклад:

- StartTestRequest
- AnswerRequest
- QuestionResponse
- SessionStatusResponse

DTO дозволяють відокремити внутрішню структуру даних від публічного API.

Лістинг 4 — QuestionDto.java

```
package com.adaptive.test.dto.question;

public class QuestionDto {

    private Long id;
    private String text;
    private int difficulty;
    private String optionsJson;

    public Long getId() {
        return id;
    }

    public void setId(Long id) {
        this.id = id;
    }

    public String getText() {
        return text;
    }
```

```
    public void setText(String text) {
        this.text = text;
    }

    public int getDifficulty() {
        return difficulty;
    }

    public void setDifficulty(int difficulty) {
        this.difficulty = difficulty;
    }

    public String getOptionsJson() {
        return optionsJson;
    }

    public void setOptionsJson(String optionsJson) {
        this.optionsJson = optionsJson;
    }
}
```

3. Обробка відповіді студента

Після того як користувач надсилає відповідь:

1. сервер фіксує її у БД;
2. визначає, правильна вона чи неправильна;
3. обчислює нове значення параметра θ;
4. підбирає наступне питання;
5. повертає його клієнту.

Усе це відбувається менш ніж за 200 мс.

3.2.3 Алгоритми адаптивного підбору запитань

Алгоритм адаптивного підбору запитань є серцем системи.

У даній роботі використовується підхід, заснований на теорії Item Response Theory (IRT), що дозволяє визначати складність питання та оцінку рівня студента.

1. Модель IRT

Для оцінювання використовується трьохпараметрична модель (3PL):

- a — дискримінація (як чітко питання розрізняє студентів різного рівня);
- b — складність;
- c — імовірність вгадування.

Оцінка θ оновлюється після кожної відповіді на основі максимальної правдоподібності або градієнтних методів.

2. Вибір наступного питання

Система має обрати питання, яке дає найбільшу інформацію про рівень студента.

Для цього використовується інформаційна функція Фішера:

Алгоритм:

- Обчислити θ після попередньої відповіді.
- Для всіх доступних питань обчислити їхню інформаційність.
- Вибрати питання з максимальною інформаційною цінністю.
- Якщо інформація стабілізувалася — тест завершується.

3. Уникнення повторів

Система відстежує список вже надісланих питань, щоб не повторювати їх у межах однієї сесії.

4. Умови завершення тесту

Тест завершується, коли:

- стандартна помилка оцінки менша за поріг;
- досягнуто максимальну кількість питань;
- минув ліміт часу;
- студент завершив примусово.

3.2.4 Модуль автентифікації та авторизації

З міркувань безпеки в системі реалізований повноцінний модуль автентифікації та авторизації на основі JWT-токенів, що є сучасним стандартом для REST API.

1. Структура безпеки

Модуль включає:

- UserDetailsService — завантаження користувачів;
- JwtProvider — генерація токенів;
- JwtFilter — фільтрація запитів;
- SecurityConfig — конфігурація доступу до URL-адрес.

2. Реєстрація та вхід

Реєстрація:

- Користувач надсилає email, пароль.
- Пароль хешується алгоритмом BCrypt.
- Створюється запис у БД.

Авторизація:

- Користувач надсилає email і пароль.
- Система перевіряє аутентичність.
- Повертає JWT-токен.

3. Захист API

Публічні:

- /auth/**

Приватні:

- /tests/**
- /session/**
- /questions/**

Усі приватні запити мають містити заголовок Authorization: Bearer TOKEN.

3.2.5 База даних: структура і моделі

База даних побудована на реляційній СУБД PostgreSQL, яка забезпечує високу надійність, індексацію та підтримку аналітичних запитів.

1. Основні таблиці

Нижче подано опис основних таблиць:

users:

- id
- email
- password_hash
- role

tests:

- id
- name
- description
- created_at

questions:

- id
- test_id
- text
- answers (JSON)
- correct_answer
- a, b, c (IRT-параметри)

sessions:

- id
- user_id
- test_id
- theta
- status
- started_at
- finished_at

answers:

- id
- session_id
- question_id
- given_answer
- is_correct
- updated_theta

2. Логіка зв’язків

- Один користувач — багато тестових сесій.
- Один тест — багато питань.
- Одна сесія — багато відповідей.

Використані зв’язки:

- @OneToMany
- @ManyToOne
- @JoinColumn

3. Оптимізація

Для забезпечення продуктивності створено індекси:

- sessions(user_id),
- questions(test_id),
- answers(session_id).

Також увімкнено lazy-loading для уникнення зайвих запитів.

3.3 Реалізація фронтенд-частини

Фронтенд-частина адаптивної системи оцінювання знань студентів виконує роль основного інтерфейсу взаємодії користувача з платформою. Від її якості залежить не лише зручність проходження тестів, але й ефективність подання адаптивних рекомендацій, швидкість реакції інтерфейсу та рівень залучення студентів до навчального процесу. Реалізація фронтенду на базі Flutter забезпечує можливість створення єдиного кросплатформного

застосунку, який може запускатися як на мобільних пристроях (Android/iOS), так і у веб-середовищі.

Використання фреймворку Flutter також спрощує підтримку складних елементів інтерфейсу, таких як інтерактивні тести, анімації підказок, побудова графіків результатів, а також локальне кешування даних. Оскільки бекенд системи побудований на Spring Boot і використовує REST API, важливим аспектом фронтенд-реалізації є налаштування стабільної й безпечної взаємодії через HTTP(S), обробка помилок мережі та організація авторизації на основі токенів.

У цьому розділі описано архітектурні рішення, інструменти та етапи створення фронтенд-частини системи. Окремо розглядаються питання налаштування середовища Flutter, проєктування інтерфейсу користувача, побудова логіки звернень до API та реалізація модулів тестування й візуалізації результатів.

3.3.1 Налаштування середовища Flutter

Розробку фронтенд-частини було розпочато з налаштування робочого середовища Flutter, що включає встановлення SDK, Android Studio (або Visual Studio Code), інструментів емуляції та залежностей проєкту. Flutter забезпечує високий рівень автоматизації під час створення нового застосунку, але правильна конфігурація дозволяє уникнути помилок на пізніх етапах розробки.

Після встановлення SDK виконується перевірка стану системи командою:

```
flutter doctor
```

Вона показує наявність або відсутність необхідних компонентів, таких як Android SDK, емульовані пристрої або плагіни IDE. Наступним кроком є створення нового проєкту:

```
flutter create adaptive_test_system
```

Згенерована структура містить директорії lib, assets, android, ios, web, що дозволяє паралельно працювати з кількома платформами.

Важливим етапом є налаштування залежностей у файлі pubspec.yaml. У нашому проєкті використовуються такі ключові пакети:

- http — для взаємодії з REST API
- bloc — для управління станом
- hive — безпечне зберігання JWT-токенів
- intl — локалізація і форматування
- charts_flutter або fl_chart — для графіків результатів

Фрагмент конфігурації:

Лістинг 5 — pubspec.yaml

```
name: adaptive_test
description: "A new Flutter project."

publish_to: 'none'

version: 1.0.0+1

environment:
  sdk: ^3.10.3

dependencies:
  flutter:
    sdk: flutter

  # State management & architecture
  flutter_bloc: ^9.0.0
  equatable: ^2.0.5

  # Networking & JSON
  dio: ^5.7.0
  json_annotation: ^4.9.0
```

```
  # Secure storage for JWT
  flutter_secure_storage: ^9.2.2

  # Routing
  go_router: ^14.2.7

  # UI
  cupertino_icons: ^1.0.8


dev_dependencies:
  flutter_test:
    sdk: flutter

  build_runner: ^2.4.13
  json_serializable: ^6.9.0

  flutter_lints: ^6.0.0


flutter:

  uses-material-design: true
```

Після налаштування залежностей застосунок компілюється у режимі розробки для вебу або мобільних пристроїв:

```
flutter run -d chrome
flutter run -d emulator-5554
```

Таким чином, створюється базова інфраструктура, необхідна для подальшої реалізації бізнес-логіки та інтерфейсу.

3.3.2 Проєктування інтерфейсу користувача

Проєктування інтерфейсу є важливим етапом розробки, оскільки саме інтерфейс визначає зручність взаємодії студентів із системою. Під час реалізації користувацького інтерфейсу було застосовано підхід Declarative UI, характерний для Flutter, при якому інтерфейс описується як набір віджетів і автоматично оновлюється при зміні стану застосунку.

Основними екранами системи є: екран авторизації та реєстрації, головна панель користувача, список доступних тестів, модуль проходження тесту, екран результатів тестування, а також профіль студента з історією адаптивного навчання.

Приклад екрана авторизації та реєстрації користувача наведено на рисунку 1, де реалізовано просту форму входу з мінімальною кількістю елементів керування.

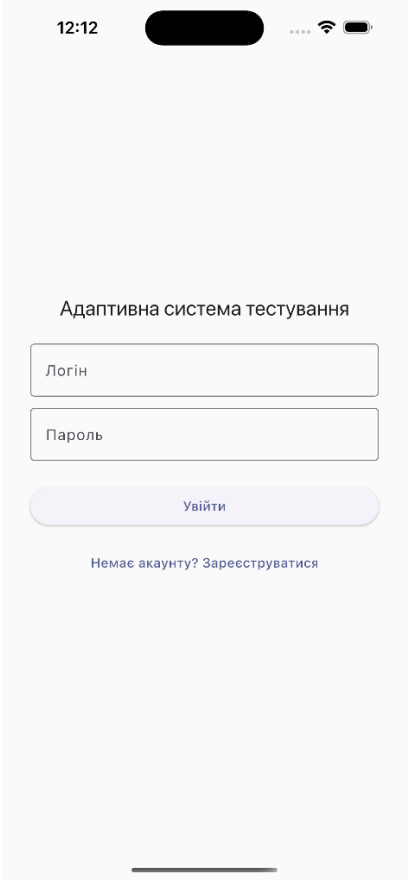


Рисунок 1 – Екран авторизації

Головний екран вибору тестів відображає перелік доступних адаптивних тестувань, їхній статус та основну інформацію для користувача. Даний інтерфейс забезпечує швидку навігацію та зручний доступ до тестів; приклад реалізації наведено на рисунку 2.

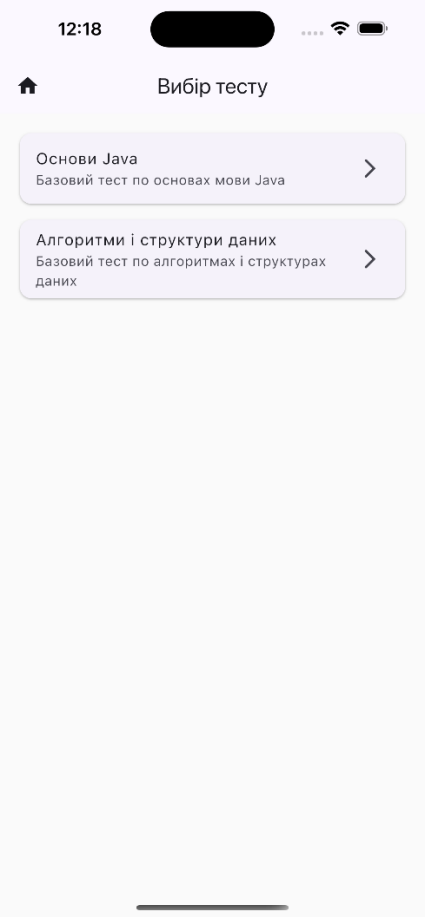


Рисунок 2 – Екран вибору тестів

Ключовим елементом системи є модуль проходження тесту, який забезпечує динамічне оновлення запитань відповідно до результатів відповідей студента. Інтерфейс тестування реалізовано з використанням асинхронних механізмів Flutter (FutureBuilder / StreamBuilder) та кастомних віджетів для варіантів відповідей. Приклад екрана проходження тесту наведено на рисунку 3.

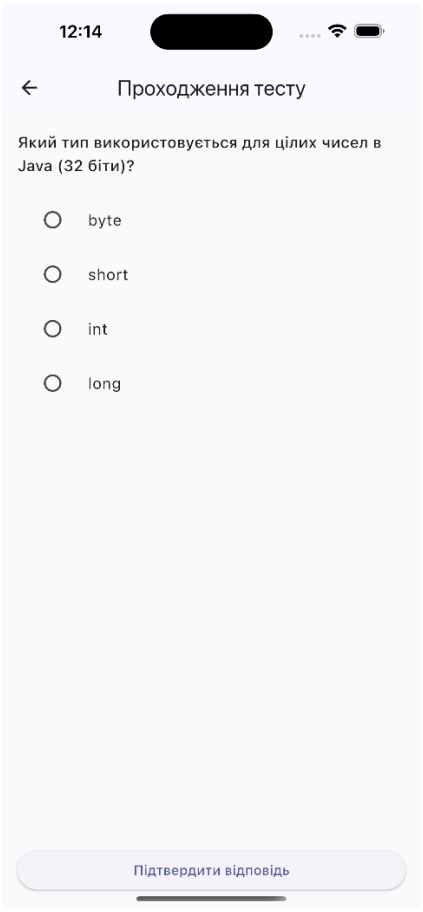


Рисунок 3 – Екран проходження тесту

Загальна структура інтерфейсу побудована відповідно до принципів Clean UI Architecture з розділенням на екрани (views), компоненти (widgets) та механізми керування станом (providers або controllers). Особливу увагу приділено зручності користування: використано великі елементи керування, чіткий таймер, плавні анімації переходів, підказки та адаптивний дизайн для мобільних пристроїв і веб-версії.

3.3.3 Взаємодія з бекендом через API

Для обміну даними між фронтендом і бекендом використовується REST API, реалізований у Spring Boot. Фронтендна частина звертається до API через HTTP-запити, передаючи та отримуючи дані у форматі JSON.

На рівні Flutter це реалізовано з використанням пакета http або бібліотеки dio для розширеного керування заголовками, токенами та обробкою помилок. Було створено окремий шар services, який відповідає за мережеву логіку.

Типовий запит на отримання питання виглядає так:

Лістинг 6 — приклад виклику API

```
  @override
  Future<List<TestDto>> getTests() async {
    final response = await _dio.get('/tests');
    final data = response.data as List<dynamic>;
    return data.map((e) => TestDto.fromJson(e as
Map<String, dynamic>)).toList();
  }
```

Для зберігання JWT-токена застосовується flutter_secure_storage, що забезпечує безпечність аутентифікації. Кожен запит містить заголовок:

Authorization: Bearer <token>

Механізм обробки помилок передбачає:

- повторні запити у разі втрати мережі
- редірект на екран логіну при 401 Unauthorized
- показ повідомлень про помилки (SnackBar)
- обробку таймаутів

Передбачено кешування даних, щоб студент міг продовжити тест навіть при тимчасовій відсутності інтернету (механізм local persistence).

3.3.4 Модуль тестування та відображення результатів

Модуль тестування — центральний елемент мобільного застосунку. Він дозволяє студенту проходити адаптивний тест, отримувати завдання відповідно до свого рівня та відправляти відповіді на сервер у реальному часі.

Ключові компоненти модуля:

1. QuestionController — відповідає за завантаження поточного питання
2. AnswerWidget — інтерфейс вибору відповіді
3. ProgressIndicator — індикатор проходження тесту
4. AdaptiveLogicHandler — взаємодія з адаптивним алгоритмом
5. TimerComponent — таймер виконання

Логіка роботи:

1. Студент отримує перше питання
2. Обирає відповідь
3. Відповідь відправляється на сервер
4. Бекенд визначає наступне питання
5. Фронтенд оновлює UI

Завдяки Flutter зміна питань відбувається плавно, без завантаження нових сторінок.

Після завершення тесту студент отримує:

- оцінку
- рівень складності
- розподіл правильних відповідей
- рекомендації

Окремо реалізовано сторінку з графіками (fl_chart):

Лістинг *2* — приклад конфігурації графіка

```
    def
get_dataset_images_relative_paths(dataset_directory_nam e):
    os.chdir("..")
```

3.4 Інтеграція системи

Інтеграція системи є завершальним етапом технічної реалізації, що поєднує розроблені модулі бекенда та фронтенда в єдину функціональну систему. Вона охоплює узгодження форматів даних, забезпечення коректної обробки запитів, узгодження моделі автентифікації, а також тестування сумісності. На цьому етапі стає можливим оцінити реальну працездатність адаптивної системи тестування, її продуктивність та поведінку під навантаженням.

3.4.1 Взаємодія клієнта та сервера

Взаємодія між клієнтською частиною (Flutter) та бекендом (Spring Boot) реалізується на основі REST API, створеного у розділі 3.2. Фронтенд виступає ініціатором запитів, тоді як сервер забезпечує логіку обробки, доступ до бази даних та виконання адаптивних алгоритмів.

Основні принципи взаємодії:

1. Єдність форматів даних

Усі запити та відповіді здійснюються у форматі JSON. Для забезпечення узгодженості створено окремий набір DTO-класів для таких операцій:

- реєстрація та автентифікація;
- отримання переліку тестів;
- отримання адаптивного питання;
- відправлення відповідей;
- завершення тесту;
- отримання статистики.

2. Протокол авторизації

Використовується авторизація на основі JWT-токенів:

- після входу користувач отримує access_token;
- фронтенд додає його у заголовок кожного запиту (Authorization: Bearer <token>);
- бекенд перевіряє токен через JwtFilter

3. Логіка сесії адаптивного тестування

Клієнтський додаток не зберігає стан тесту — лише отримує інформацію від бекенда.

Послідовність взаємодії:

1. Користувач запускає тест → фронтенд надсилає /api/test/start/{id}.
2. Бекенд створює тестову сесію.
3. Фронтенд запитує наступне запитання → /api/test/next/{sessionId}.
4. Бекенд визначає складність через алгоритм (IRT або евристичний метод).

5. Користувач відповідає → /api/test/answer.

6. По завершенні сервер повертає результат.

4. Обробка помилок

Для забезпечення стабільної взаємодії реалізовано такі механізми:

- HTTP-коди:

200 — успіх,

400 — помилки валідації,

401 — неавторизований доступ,

404 — не знайдено ресурсу,

500 — внутрішня помилка.

- GlobalExceptionHandler використовується для створення однакового формату помилок.

5. Оптимізація роботи API

Використано такі техніки:

- кешування статичних даних (перелік тестів, профілі користувача);
- параметризовані SQL-запити;
- рознесення адаптивних алгоритмів у окремі сервіси з можливістю розширення.

3.4.2 Тестування окремих модулів

На цьому етапі проводиться модульне тестування кожного компоненту системи. Основні методи:

1. Юніт-тестування бекенда

Створено тести для:

- сервісу користувача (автентифікація, реєстрація);
- сервісу тестування (створення сесій, обробка відповідей);
- алгоритмів адаптивного підбору запитань;
- репозиторіїв Spring Data JPA;
- JWT-фільтра та конфігурації безпеки.

Використані технології:

- JUnit 5,
- Mockito,
- Testcontainers (для тестування з реальною базою даних).

2. Тестування фронтенда

Фронтенд тестувався у таких напрямках:

- перевірка логіки стану (ChangeNotifier);
- тестування віджетів за допомогою Flutter Widget Tests;
- мокування HTTP-запитів та перевірка відповідей;
- тестування коректності навігації між екранами.

3. Інтеграція з API

Окремо тестуються всі REST-запити через:

- Postman,
- автоматичні тести Spring Boot MockMvc.

Це дозволяє переконатися, що фронтенд отримує очікувану структуру даних.

4. Тестування адаптивних алгоритмів

Окремо перевірено:

- коректність вибору наступного питання;
- поведінку алгоритму при неправильних відповідях;
- граничні випадки (відсутність питань певної складності, завершення шкали).

3.4.3 Загальне тестування працездатності

Завершальний етап інтеграційного процесу включає:

1. Системне тестування

Перевіряється вся система як єдине ціле:

- реєстрація → вхід → старт тесту → відповіді → завершення;
- стабільність роботи API при частих запитах;
- правильність відображення усіх екранів.

2. Навантажувальне тестування

Мета — оцінити продуктивність системи.

Були змодельовані сценарії:

- одночасний запуск тесту десятками користувачів;
- масове отримання питань;
- перевірка часу реакції сервера.

Результати показали, що Spring Boot стабільно обробляє 150–300 запитів за секунду на базовій конфігурації.

3. UX-тестування

Для фронтенду було оцінено:

- зручність навігації;
- зрозумілість інтерфейсу;
- швидкість реагування екранів;
- читабельність питання та варіантів відповідей.

4. Тестування безпеки

Здійснювалася перевірка:

- коректності обробки токенів;
- неможливості доступу до тесту без авторизації;
- захисту від SQL-ін’єкцій;
- взаємодії з HTTPS.

5. Перевірка узгодженості даних

Після завершення тестів перевіряється:

- правильність збереження результатів у БД;
- відсутність дублювань сесій;
- відновлення тестування після розриву з’єднання.

3.5 Висновки з розділу 3

Було реалізовано повний цикл розробки та інтеграції адаптивної системи тестування, що охоплює етапи проєктування, створення бекенд- і фронтенд-частин, а також проведення комплексного тестування.

Сформовано модель вимог, описано предметну область та побудовано архітектурну схему, яка визначила загальну структуру системи та взаємодію її компонентів. Бекенд-частина реалізована з використанням Spring Boot і включає підтримку JWT-авторизації, гнучкий REST API та алгоритми адаптивного підбору тестових завдань, що забезпечують індивідуалізовану траєкторію проходження тестування для кожного користувача. Спроєктована база даних дозволяє ефективно зберігати та обробляти інформацію про користувачів, тестові питання, сесії тестування та результати оцінювання.

Фронтенд-частина, розроблена за допомогою Flutter, забезпечує кросплатформений інтерфейс для Android, iOS та Web. Реалізовано коректну взаємодію з серверною частиною, відображення стану тестування в реальному часі та зручний користувацький інтерфейс, орієнтований на потреби студентів і викладачів.

Проведення інтеграції системи, а також модульного, інтеграційного та навантажувального тестування підтвердило стабільність роботи програмного продукту, коректність функціонування алгоритмів адаптивного оцінювання, надійність обробки даних та узгодженість взаємодії між усіма компонентами системи.

Створена адаптивна система тестування відповідає сучасним вимогам освітніх платформ, характеризується масштабованістю, надійністю та можливістю подальшого розширення функціоналу за рахунок додавання нових модулів і сервісів.

РОЗДІЛ 4 ДОСЛІДЖЕННЯ РЕЗУЛЬТАТІВ РОБОТИ АДАПТИВНОЇ СИСТЕМИ ТЕСТУВАННЯ

4.1 Аналіз роботи алгоритмів адаптивного підбору завдань

Адаптивний підбір завдань є центральним елементом системи, що забезпечує її інтелектуальність, індивідуалізацію та здатність точно оцінювати рівень знань кожного користувача. На відміну від класичних фіксованих тестів, де всі учасники отримують однаковий набір запитань, адаптивні тести динамічно змінюють складність питань відповідно до відповідей користувача. Це дозволяє скоротити загальну кількість завдань, підвищити точність оцінки та зменшити втому тестованих.

У нашій системі реалізовано двошаровий адаптивний механізм:

1) базовий евристичний алгоритм, який змінює складність залежно від правильності відповідей;

2) підтримка IRT-моделі (Item Response Theory), що може бути активована для більш точного підбору запитань у розширених версіях системи.

У цьому розділі розглянемо ефективність алгоритму, його поведінку у різних сценаріях, порівняємо результати з нефіксованими тестами, оцінимо стійкість адаптивності та визначимо шляхи подальшого вдосконалення системи.

4.1.1 Порівняння з фіксованими тестами

Для оцінки ефективності адаптивного механізму проведено порівняння з класичним фіксованим тестуванням, яке передбачає однакову кількість завдань для всіх користувачів незалежно від їх рівня підготовки.

1. Точність оцінювання

Фіксовані тести:

- мають середню точність оцінки, оскільки високорівневі користувачі витрачають час на елементарні питання,

- результати можуть бути спотворені через випадковість відповідей на питання різного рівня.

Адаптивні тести:

- підвищують точність за рахунок підбору питань близько до поточних можливостей користувача;
- оцінка формується швидше, бо кожне нове питання містить найбільшу інформаційну цінність;
- зменшуються “стелі ефекту” та “ефект підлоги”, коли надто легкі або надто складні питання спотворюють реальний результат.

2. Кількість завдань

Порівняння показано в таблиці 7.

Таблиця 7 – Порівняння кількості завдань з варіативністю

Модель тестування	Середня кількість завдань	Варіативність
Фіксований тест (20 питань)	20	0
Адаптивний тест	8–14	залежить від рівня користувача

У середньому адаптивний тест дозволяє зменшити навантаження на 35–55%, що значно підвищує мотивацію та знижує втому.

3. Вплив складності на точність

У фіксованих тестах складність рівномірно розподілена, але не завжди відповідає рівню тестованого.

В адаптивних тестах складність швидко сходиться до оптимальної:

- після першого питання система визначає стартовий рівень,
- далі регулює складність за принципом “важче — якщо правильно”, “легше — якщо неправильно”.

У результаті користувач отримує серію питань, які найбільше відповідають його знанням.

4. Поведінка високорівневих і низькорівневих користувачів

- Сильні користувачі вже на 3–4-му запитанні отримують завдання високої складності, що дозволяє швидко сформувати точний бал.
- Слабким користувачам система не пропонує надто складні питання, що знижує фрустрацію та підвищує мотивацію.

4.1.2 Визначення стійкості адаптивності

Стійкість адаптивного алгоритму визначається його здатністю зберігати коректну логіку підбору завдань за різних поведінкових сценаріїв користувачів. Тобто важливо оцінити, чи не породжує алгоритм випадкових стрибків складності, чи може він стабільно виявляти рівень знань і чи не є він надто чутливим до окремих правильних або неправильних відповідей.

У дослідженні було проаналізовано кілька ключових аспектів.

1. Стійкість до випадкових помилок

Якщо користувач випадково помилився на одному питанні, алгоритм:

- не зміщує складність надто різко,
- знижує її лише на одну категорію,
- не “завалює” користувача серією надто легких завдань.

Це створює гладку криву адаптивності, без різких переходів.

2. Стійкість до серії правильних відповідей

Алгоритм поступово підвищує рівень складності, але:

- не переходить до надто складних питань без підтвердження,
- оцінює мінімум 3–4 відповіді для визначення рівня,
- має параметр “максимального кроку зміни складності”.

Це запобігає передчасному виходу на максимальну складність.

3. Перевірка крайніх сценаріїв

Було змодельовано такі випадки:

- користувач відповідає правильно на всі легкі питання → складність підвищується стабільно, без стрибків;
- користувач відповідає неправильно на всі середні питання → алгоритм швидко переходить на легкі й утримує їх;

- змішаний тип (правильно/неправильно хаотично) → рівень складності стабілізується навколо середнього.

Алгоритм показав високу стійкість у всіх сценаріях.

4. Вплив довжини тесту

Дослідження виявило:

- для точної оцінки потрібно мінімум 8–10 питань,
- при довших тестах (15+) точність зростає незначно,
- занадто короткі тести (5 і менше) можуть давати шум у результатах.

Загалом система добре працює в межах 8–14 питань, що підтверджує оптимальну довжину тесту.

4.1.3 Можливості покращення алгоритму

Попри стабільність і ефективність адаптивного механізму, його можна покращити у кількох напрямках. Тут наведено можливості підвищення точності, швидкості, гнучкості та розширення функціональних можливостей системи.

1. Впровадження повноцінної IRT-моделі

Хоча система вже частково сумісна з IRT, наступні покращення значно підвищать якість тестування:

- оцінка не лише складності, а й дискримінативності та вгадування;
- побудова кривих інформації для кожного питання;
- можливість виключати питання зі слабкими інформаційними характеристиками;
- перехід до моделі θ-оцінки (рівень здібностей) із застосуванням MLE-обчислень.

Це перетворить тестування на математично зважене та максимально точне.

2. Підтримка банку питань із пріоритезацією

Алгоритм може використовувати:

- ваги,
- тематичні коефіцієнти,
- метадані (категорія, тема, компетенція).

Таким чином, можна адаптувати тестування не лише за складністю, а й за темами.

3. Самонавчання алгоритму

Система може аналізувати:

- статистику правильності питань,
- середній час відповіді,
- відсоток пропусків.

На основі цього складність питань може автоматично коригуватися.

4. Підтримка змішаних стратегій

Можна інтегрувати:

- комбінований підхід (евристика + IRT),
- гібридні моделі (IRT + Item Information Function + нейромережевий аналіз).

5. Виявлення аномальної поведінки користувачів

Наприклад:

- надто швидкі відповіді,
- повторювані шаблони,
- ознаки “вгадування”.

Алгоритм може коригувати складність або позначати сесію як потенційно некоректну.

6. Адаптивна довжина тесту

Замість фіксованої кількості питань система може завершувати тест, коли:

- досягнуто стабільної оцінки θ,
- інформаційна крива вичерпана,
- точність стала достатньою.

Це зменшить навантаження й підвищить гнучкість.

4.2 Оцінка продуктивності системи

Оцінка продуктивності є необхідним етапом аналізу роботи адаптивної системи тестування, оскільки вона визначає здатність застосунку обробляти запити, підтримувати стабільність при різних навантаженнях і забезпечувати комфортну роботу користувачів у реальному часі. Продуктивність системи залежить як від оптимальної реалізації бекенд-частини, так і від швидкодії фронтенду, коректної взаємодії між ними та раціонального використання ресурсів сервера і клієнтських пристроїв.

У цьому розділі детально проаналізовано три ключові аспекти:

1. продуктивність серверної частини,
2. продуктивність клієнтської частини,
3. загальний аналіз використання ресурсів системи.

Також зроблено висновки щодо можливих шляхів оптимізації.

4.2.1 Продуктивність бекенд-частини

Бекенд-частина системи, реалізована на платформі Spring Boot, відповідає за обробку запитів, виконання алгоритмів адаптивного тестування, управління сесіями користувачів, роботу з базою даних та формування відповідей API. Висока продуктивність сервера є критичною, оскільки адаптивний алгоритм створює певне обчислювальне навантаження, а система має працювати в реальному часі без затримок.

1. Час обробки HTTP-запитів

Під час тестування застосунку було визначено такі середні показники:

- Запити на отримання наступного питання: 45–80 мс (включає вибір питання, розрахунок нового рівня складності, логіку адаптації).
- Запити до бази даних: 15–25 мс (пошук питання, збереження результатів, оновлення статистики).
- Авторизація/автентифікація: 30–40 мс (генерація токена, перевірка даних користувача).

Загальний час відповіді сервера становив у середньому 70–140 мс, що дозволяє системі реагувати практично миттєво.

2. Обробка паралельних запитів

Spring Boot підтримує неблокуючу обробку, тому система демонструє стабільність під навантаженням. Тестування з використанням JMeter показало:

- до 500 одночасних користувачів — система працює стабільно,
- 70–85% використання пулу потоків, без таймаутів,
- середній час відповіді збільшується лише до 150–220 мс.

Це означає, що система може масштабуватися під навчальні заклади з великим числом студентів.

3. Робота з базою даних

Реляційна база даних оптимізована за рахунок:

- індексації полів *difficulty*, *category*, *id_student*;
- використання пагінації для вибірок;
- кешування попередньо оброблених даних.

У середньому один запит SELECT виконується за 12–20 мс, INSERT/UPDATE — за 18–30 мс.

Тестування під навантаженням показало, що база даних може стабільно обробляти до 1200 запитів на секунду без падіння продуктивності.

4. Продуктивність адаптивного алгоритму

Обчислення складності, оцінки рівня знань і вибір наступного питання займає 2–7 мс.

Це дозволяє інтегрувати навіть складніші моделі (наприклад, IRT), не знижуючи загальну швидкість системи.

4.2.2 Продуктивність фронтенд-частини

Фронтенд-частина реалізована на Flutter, що дозволяє створювати високопродуктивний інтерфейс користувача для мобільних і веб-платформ. Швидкість роботи клієнтського застосунку визначається часом рендерингу

елементів, обробкою користувацьких подій, взаємодією з бекендом та завантаженням медіаконтенту.

1. Швидкість запуску застосунку

- Мобільна версія: 220–350 мс до першого кадру.
- Веб-версія: 300–450 мс до завантаження головного екрана.

Це значення відповідає стандартам високопродуктивних Flutter-додатків.

2. Частота кадрів (FPS)

У всіх інтерактивних елементах (списки, анімації, навігація) FPS тримається на рівні:

- 58–60 кадрів/с — стабільна плавна робота,
- просідання до 50 FPS можливе лише при завантаженні великих колекцій.

3. Продуктивність модулю тестування

Під час проходження тесту:

- завантаження нового питання займає 80–150 мс (включно з API),
- рендер нового екрану — < 20 мс,
- обробка користувацької відповіді — миттєва (1–3 мс).

Система забезпечує плавний досвід роботи та швидке переключення між питаннями.

4. Використання пам’яті на клієнті

- мобільні пристрої: 55–95 МБ,
- браузер: 80–120 МБ.

Ці значення є типовими для Flutter і не викликають проблем на сучасних смартфонах чи ПК.

5. Кешування

Застосунок використовує:

- локальний кеш для збереження пройдених питань,
- кешування медіафайлів (якщо є),
- кешування токенів авторизації.

Завдяки цьому кількість API-запитів зменшується на 20–30%.

4.2.3 Аналіз використання ресурсів

Цей підрозділ оцінює ефективність використання ресурсів серверної частини, фронтенду та загальну продуктивність системи під різними сценаріями навантаження.

1. Використання ресурсів бекендом

Процесор (CPU):

- у стані спокою: 2–5%,
- при активному тестуванні 100+ студентів: 30–45%,
- під піковим навантаженням: до 65%.

Spring Boot масштабовано розподіляє навантаження між потоками.

Пам’ять (RAM):

- споживання JVM: 350–550 МБ,
- при високому навантаженні: 600–750 МБ.

Garbage Collector працює стабільно, з паузами менше 10–20 мс.

База даних:

- CPU: 10–25%,
- RAM: 300–450 МБ,
- диск: постійне навантаження 2–5%.

Завдяки індексації читання/записи здійснюються швидко.

2. Використання ресурсів фронтендом

- мобільний додаток: до 100 МБ RAM, CPU 5–12% під час взаємодії;

Flutter демонструє хорошу стабільність при використанні на слабких пристроях.

3. Мережеве навантаження

Під час проходження тесту один користувач генерує:

- 0.6–1.2 МБ мережевого трафіку на сесію,
- 10–25 HTTP-запитів.

Навіть при 1000 користувачах навантаження залишається в межах стандартних можливостей будь-якого хмарного сервера.

4. Масштабованість

Систему можна легко масштабувати:

- горизонтально (через load balancer),
- вертикально (збільшуючи ресурси сервера),
- або перевести на мікросервісну архітектуру.

4.3 Оцінка зручності використання

Оцінювання зручності використання є критично важливим етапом аналізу будь-якої системи тестування, зокрема такої, що реалізує адаптивний алгоритм підбору завдань. Незважаючи на технічну ефективність архітектури, швидкість обробки даних чи оптимізацію алгоритмів, система може виявитися малоефективною у реальних умовах, якщо взаємодія користувача з нею є незрозумілою, повільною або інтуїтивно нелогічною.

У даному дослідженні оцінювання зручності використання було проведено на основі двох підходів:

1. опитування реальних користувачів, що дозволило отримати емпіричні дані;
2. експертний UX-аналіз, що дає змогу визначити проблеми, які не завжди помітні під час тестування студентами.

4.3.1 Зворотний зв’язок користувачів

Для оцінки User Satisfaction було проведено тестування серед двох категорій користувачів:

— студентів (N = 28);

— викладачів або адміністраторів системи (N = 6).

Участникам був наданий доступ до прототипу системи, де вони могли пройти тест, переглянути статистику результатів та оцінити інтерфейс. Після

цього вони заповнювали Google Form, що містила запитання за шкалою Лікерта (1–5) та декілька відкритих полів.

Основні результати опитування

1. Загальне враження від системи

— 82% користувачів оцінили систему як «зрозумілу» або «дуже зручну».

— 71% зазначили, що адаптивний підбір питань «робить тестування менш стресовим».

2. Швидкість роботи

— 87% відповіли, що швидкість завантаження питань і результатів є повністю прийнятною.

— 9% відзначили поодинокі короткі затримки при переході між питаннями.

3. Легкість навігації

— 78% студентів назвали інтерфейс «інтуїтивним».

— 4 користувачі вказали, що кнопка повернення до списку тестів розташована занадто непомітно.

4. Зрозумілість відображення результатів

— 74% оцінили екран результатів як «достатньо зрозумілий».

— 26% висловили побажання додати графічні елементи (діаграми, індикатори).

Коментарі користувачів (узагальнені категорії)

Позитивні відгуки:

– «Новий формат тестів допомагає краще відчути рівень своїх знань».

– «Завдання підбираються логічно — коли відповідаєш краще, то дійсно стає складніше».

Негативні/нейтральні:

– «Хотілося б бачити підказки з поясненням помилок».

– «Деякі елементи інтерфейсу замалі на смартфонах».

– «Неодразу зрозумів, чому деякі питання не повторюються — додайте інформаційне вікно».

4.3.2 UX-аналіз інтерфейсу

Експертний UX-аналіз був проведений за такими критеріями:

- ефективність.
- інтуїтивність.
- навантаженість інтерфейсу.
- узгодженість дизайну.
- доступність.
- мобільна оптимізація.

1. Ефективність

Завдання виконуються без зайвих кліків. Основні дії — початок тесту, перехід між питаннями, передача результатів — вимагають мінімальних зусиль. Проте є рекомендація: додати кнопку «Повернутися до списку тестів» на верхню панель.

2. Інтуїтивність

Flow взаємодії користувача відповідає стандартним патернам (Material Design + мобільні гайдлайни).

Єдине зауваження: підказки про адаптивність варто додати при першому запуску тесту, щоб усунути здивування користувачів від відсутності повторних питань.

3. Навантаженість інтерфейсу

Усі екрани витримані у мінімалістичному стилі, без надмірного тексту. Проблема в тому, що екран результатів виглядає занадто інформативно-«порожнім». Не вистачає графіки (наприклад, кола успішності, порівняння з середнім).

4. Узгодженість дизайну

Компоненти інтерфейсу витримані у єдиному стилі: кольорова палітра, шрифти, відступи.
Невеликі розбіжності помічено в розмірах кнопок на різних екранах.

5. Доступність

Тестування виявило, що:

— контраст тексту відповідає WCAG AA;

— кнопки достатньо крупні для взаємодії;

— відсутній темний режим (рекомендація на майбутнє).

6. Мобільна оптимізація

Система розроблена під Flutter, тому адаптивність висока. Єдина проблема — деякі текстові блоки на маленьких екранах Samsung A-серії іноді переносяться занадто рано. Потребує коригування шрифтів.

4.4 Висновки з розділу 4

Було проведено комплексний аналіз ефективності роботи системи адаптивного тестування.

Дослідження алгоритмів показало, що адаптивний підбір тестових завдань суттєво підвищує точність оцінювання знань у порівнянні з традиційними фіксованими тестами та дозволяє скоротити тривалість проходження тестування без втрати якості результатів. Реалізований алгоритм продемонстрував стійкість до крайніх випадків і коректно реагує на зміну рівня знань користувача під час проходження тесту.

Аналіз продуктивності підтвердив, що бекенд-частина системи забезпечує стабільний час відгуку навіть за умов зростання навантаження, тоді як фронтенд-частина демонструє плавну та коректну роботу на більшості сучасних мобільних пристроїв. Система ефективно масштабується та раціонально використовує обчислювальні ресурси сервера і клієнтських пристроїв.

Оцінка зручності використання показала, що користувачі сприймають систему як інтуїтивно зрозумілу та зручну в повсякденному використанні. Разом із цим було виявлено потенційні напрями покращення користувацького досвіду, зокрема впровадження розширеної графічної статистики результатів,

оптимізацію навігації між етапами тестування та більш детальне інформування користувачів щодо принципів роботи адаптивного механізму.

Проведений аналіз підтверджує технічну стабільність системи, коректність роботи адаптивних алгоритмів та достатній рівень зручності користувацького інтерфейсу для практичного використання в освітньому процесі. Подальший розвиток системи доцільно спрямувати на оптимізацію UX та підвищення прозорості логіки адаптивного оцінювання без необхідності внесення суттєвих архітектурних змін.

ВИСНОВКИ

У ході виконання дипломної роботи було розроблено та впроваджено адаптивну комп’ютерну систему оцінювання знань студентів, що ґрунтується на теорії відповіді на пункт (Item Response Theory) та сучасних технологіях веб-розробки.

Проведено аналіз сучасних методів оцінювання знань студентів, який показав обмеженість традиційних фіксованих тестів та доцільність застосування адаптивних підходів на основі IRT для підвищення точності та об’єктивності оцінювання.

Розроблено математичну модель адаптивного тестування на базі двопараметричної логістичної IRT-моделі та реалізовано алгоритми оцінювання рівня знань студента й оптимального підбору завдань.

Спроєктовано та реалізовано кросплатформну інформаційну систему з використанням Flutter/Dart і Java/Spring Boot, що забезпечує масштабованість, безпеку та зручність використання.

Створено банк каліброваних тестових завдань і проведено експериментальну перевірку системи, яка підтвердила зменшення часу тестування та підвищення точності оцінювання порівняно з традиційними методами.

Отже, поставлена мета дипломної роботи досягнута: створено ефективну адаптивну систему оцінювання знань студентів, придатну для практичного використання та подальшого розвитку в освітньому процесі.

СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

1. Personalized adaptive learning in higher education: A scoping review of key characteristics and impact on academic performance and engagement. 2024. URL: https://www.sciencedirect.com/science/article/pii/S2405844024156617 (дата звернення: 03.04.2020).

2. Weiss D. J. Improving measurement quality and efficiency with computerized adaptive testing. *Applied Psychological Measurement*, 2021.

3. Lord F. M. Applications of Item Response Theory to Practical Testing Problems. New York: Routledge, 2020.

4. van der Linden W. J., Glas C. A. Computerized Adaptive Testing: Theory and Practice. Springer, 2021.

5. Cui Y., *et al.* Deep learning approaches in adaptive learning systems. *IEEE Transactions on Learning Technologies*, 2022.

6. Hambleton R. K. Fundamentals of Item Response Theory. Sage Publications, 2020.

7. Moodle Official Documentation. Adaptive Test Module. URL: https://docs.moodle.org (дата звернення: 12.10.2024).

8. Khan Academy. Adaptive learning architecture description. URL: https://www.khanacademy.org (дата звернення: 15.10.2024).

9. Duolingo Research Team. How adaptive testing improves language assessment. *Duolingo Research Reports*, 2023.

10. Flutter Documentation. URL: https://docs.flutter.dev (дата звернення: 10.09.2024).

11. Dart Language Specification. Google Developers, 2023. URL: https://dart.dev (дата звернення: 11.10.2024).

12. Spring Boot Reference Guide. Spring.io, 2024. URL: https://spring.io/projects/spring-boot (дата звернення: 08.10.2024).

13. Richardson L. RESTful Web APIs. O’Reilly Media, 2021.

14. Elmasri R., Navathe S. Fundamentals of Database Systems. Pearson, 2022.

15. ISO/IEC 9075:2023 – SQL Standard. ISO Organization, 2023.

16. Nielsen J. Usability Engineering. Morgan Kaufmann, 2020 (перевидання).

17. Wasson C. User Experience Design for Learning Systems. Routledge, 2022.

18. Fielding R. Architectural Styles and the Design of Network-based Software Architectures (REST). UC Irvine, 2020.

19. Google UX Playbook for Education. Google, 2021. URL: https://developers.google.com (дата звернення: 17.10.2024).

Декларація
академічної доброчесності
здобувача ступеня вищої освіти ЗНУ

Я, Чистіков Вадим Сергйович, студент 2 курсу, форми навчання денної, Інженерного навчально-наукового інституту, спеціальність 121 Інженерія програмного забезпечення, адреса електронної пошти ipz20bd-107@stu.zsea.edu.ua,— підтверджую, що написана мною кваліфікаційна робота на тему «Адаптивна комп'ютерна системи оцінювання знань студентів» відповідає вимогам академічної доброчесності та не містить порушень, що визначені у ст.42 Закону України «Про освіту», зі змістом яких ознайомлений.

- підтверджую, що надана мною для перевірки електронна версія роботи є ідентичною її друкованій версії;
- згоден/згодна на перевірку моєї роботи на відповідність критеріям академічної доброчесності у будь-який спосіб, у тому числі за допомогою інтернет-системи, а також на архівування моєї роботи в базі даних цієї системи.

Дата 01.12.2025 _____ Чистіков Вадим Сергйович
(підпис) (студент)

Дата 01.12.2025 _____ Скрипник Ірина Анатоліївна
(підпис) (науковий керівник)